5-2010

# An Exploratory Study on Issues and Challenges of Agile Software Development with Scrum

Juyun Joey Cho
*Utah State University*

# AN EXPLORATORY STUDY ON ISSUES AND CHALLENGES OF AGILE SOFTWARE DEVELOPMENT WITH SCRUM

by

Juyun Joey Cho

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

DOCTOR OF PHILOSOPHY

in

Education
(Management Information Systems)

Approved:

| | |
|---|---|
| _____ | _____ |
| Dr. David H. Olsen<br>Major Professor | Dr. Jeffery Johnson<br>Committee Member |
| _____ | _____ |
| Dr. John D. Johnson<br>Committee Member | Dr. Sherry Marx<br>Committee Member |
| _____ | _____ |
| Dr. Karina Hauser<br>Committee Member | Dr. Byron Burnham<br>Dean of Graduate Studies |

UTAH STATE UNIVERSITY
Logan, Utah

2010

# ABSTRACT

An Exploratory Study on Issues and Challenges of Agile Software Development with Scrum

by

Juyun Joey Cho, Doctor of Philosophy

Utah State University, 2010

Major Professor: Dr. David H. Olsen
Department: Management Information Systems

The purpose of this dissertation was to explore critical issues and challenges that might arise in agile software development processes with Scrum. It also sought to provide management guidelines to help organizations avoid and overcome barriers in adopting the Scrum method as a future software development method. A qualitative research method design was used to capture the knowledge of practitioners and scrutinize the Scrum software development process in its natural settings. An in-depth case study was conducted in two organizations where the Scrum method was fully integrated in every aspect of two organizations' software development processes. One organization provides large-scale and mission-critical applications and the other provides small- and medium-scale applications. Differences between two organizations provided useful contrasts for the data analysis.

Data were collected through an email survey, observations, documents, and semi-structured face-to-face interviews. The email survey was used to refine interview

questions; all of the interviews were audio-taped and transcribed, and later coded for analysis. Triangulation in the data collection process provided useful information for different perspectives on the issues, allowed for cross-checking, and yielded stronger substantiation of concepts and common categories.

In the first round of data analysis, an open coding technique was used to identify possible concepts, along with their properties and dimensions. The open coding technique is a form of content analysis where the data are read and categorized into concepts. In the second round, the codes were reviewed, and the concepts were organized by recurring themes. These themes were used later as a basis for creating a set of stable and common categories. The final stage of data analysis was completed through axial coding, which depends on a synthetic technique of making connections between categories and sub-categories to build a more comprehensive scheme. In the process of data analysis, grounded theory was employed with the aim of generating descriptive and explanatory theory associated with an agile software development process.

The research presented four common categories of issues and challenges of the Scrum method, and management guidelines to help organizations that are already using the Scrum method or planning to employ it in the future. The framework for a hybrid software development model is then proposed as a future study.

(236 pages)

# ACKNOWLEDGEMENTS

I wish to acknowledge the many people who have encouraged and supported me during my work on the doctoral program and on this dissertation. First, I wish to acknowledge the insightful suggestions and advice of my chair, Dr. David Olsen, in the recent push to completion of my dissertation. Dr. Sherry Marx was instrumental in exposing me to much of the essential research methodology and played a key role in helping me establish the foundation of qualitative research skills and knowledge. Through her two qualitative research methodology classes, I was able to perform a pilot study and refine my initial research ideas and approach. The rest of my committee also shared with me their unique insights from their particular fields of specialization - Dr. John Johnson, Dr. Jeff Johnson, and Dr. Karina Hauser.

I wish to acknowledge the feedback and support from my colleagues at Colorado State University at Pueblo (CSUP) and many other friends who have enriched my thinking and invigorated my intellectual curiosity by sharing their different interdisciplinary perspectives. I particularly want to mention Dr. Rick Huff at CSUP, who offered unparalleled professional expertise and invaluable suggestions on my dissertation. His contribution to the final stage of my dissertation was enormously helpful. I also thank Dr. Jean Pratt at University of Milwaukee - Eau Claire, who initially led me to a Ph.D. program and the academic world and remained as a role model in the teaching and research field.

I wish to acknowledge two firms who allowed me to pursue my field research in their software development environments. Particularly, I want to thank Dr. Dennis

Phillips, who arranged research resources in his firm and facilitated surveys and interview questions. I also want to thank all the development professionals and my co-workers, who shared with me their expertise, insights, and precious time.

I acknowledge my parents, who raised me in good family environment and tried to provide the best possible educational opportunity. Despite the immense geographical distance that separate us, they have offered loving encouragement and support throughout my academic years. My brother and sister were also supportive and took a huge responsibility off of me in taking care of my mother, who passed away last year.

I finally acknowledge my wife and eternal soul mate, Dr. Inhae Kim, who encouraged me to get into a Ph.D. program after working for several years in the software industry, believed in my potential abilities, and has supported me with unconditional love and confidence. She and my daughter Megan have brought me a new dimension of life. Without their love and moral support, this dissertation would not have reached its fruition.

Juyun Joey Cho

CONTENTS

Page

ABSTRACT ..... III

ACKNOWLEDGEMENTS ..... V

LIST OF TABLES ..... X

LIST OF FIGURES ..... XII

INTRODUCTION ..... 1

The Problem ..... 1
Problem Statement ..... 4

REVIEW OF LITERATURE ..... 5

Introduction and Purpose of the Review of Literature ..... 5
Early Stage of Software Development Methods ..... 6
Traditional Software Development Methods ..... 7
Agile Software Development Methods ..... 12

Overview of Agile Methods ..... 12
Lean Manufacturing Principles and Agile Methods ..... 17
Characteristics, Strengths, Weaknesses of TSDMs and ASDMs ..... 20
Agile Methods for Large-Scale Projects ..... 21

Scrum Software Development Method ..... 24

The Philosophical Roots of Scrum ..... 24
The History and Practices of Scrum ..... 25
Empirical Process Control ..... 27
Framework of Scrum ..... 28
Flow of Scrum ..... 36

Rational Unified Process ..... 37

The Inception Phase ..... 42
The Elaboration Phase ..... 42
The Construction Phase ..... 43
The Transition Phase ..... 43

Quantitative Versus Qualitative Research Methods ..... 43

Qualitative Research Methods in Information Systems ..... 47

Case Study Research ..... 49
Action Research ..... 50
Ethnography ..... 50
Grounded Theory ..... 51

RESEARCH METHODS AND PROCEDURES ..... 52

Introduction ..... 52
Rationale for Selecting Case Study Research ..... 52
Type of Case Study Research ..... 53
Unit of Analysis in Case Study ..... 54
Site Selection ..... 55

ABC Firm ..... 56
XYZ Firm ..... 57
Comparison of Two Firms ..... 58

Data Sources ..... 58
Grounded Theory ..... 62

DATA ANALYSIS AND RESEARCH RESULTS ..... 64

The Process of Data Analysis ..... 64
ABC Firm ..... 65

Human Resource Management Factor ..... 66
Structured Development Process Factor ..... 79
Environment Factor ..... 89
Information Systems and Technology Factor ..... 94

XYZ Firm ..... 100

Human Resource Management Factor ..... 101
Structured Development Processing Factor ..... 108
Environmental Factor ..... 120
Information Systems and Technology Factor ..... 126

DISCUSSION AND MANAGEMENT GUIDELINES ..... 130

Issues and Challenges of Scrum ..... 130

Human Resource Management ..... 130
Structured Development Process ..... 135

Environment ..... 142
Information Systems and Technology ..... 147

Management Guidelines ..... 150

Guidelines for Co-located Scrum Teams ..... 151
Additional Guidelines for Geographically Distributed Scrum Teams ..... 154

A THEORETICAL MODEL, FUTURE STUDY, AND CONCLUSIONS ..... 156

A Theoretical Model ..... 156
Future Study: A Hybrid Model ..... 164
Limitations of Present Study ..... 168
Conclusions ..... 169

REFERENCES ..... 170

APPENDICES ..... 178

APPENDIX A. PRODUCT BACKLOG ..... 179
APPENDIX B. SPRINT BACKLOG ..... 181
APPENDIX C. BURNDOWN CHART ..... 183
APPENDIX D. C# CODING STANDARD ..... 185
APPENDIX E. INTERVIEW QUESTIONS ..... 200
APPENDIX F. CLASS DIAGRAM ..... 204
APPENDIX G. SEQUENCE DIAGRAM ..... 206
APPENDIX H. QUALITATIVE STUDIES ..... 208

CURRICULUM VITAE ..... 216

# LIST OF TABLES

Table Page

1. Waterfall Model Deliverables ..... 8

2. List of ASDMs ..... 13

3. Manifesto for Agile Software Development ..... 14

4. Principles Behind the Agile Manifesto ..... 14

5. Discriminator Between ASDMs and TSDMs ..... 16

6. Principles of Lean Manufacturing ..... 17

7. The Seven Wastes ..... 18

8. Characteristics, Strengths, and Weaknesses of TSDMs and ASDMs ..... 22

9. Six Principles of the New Product Development Process ..... 25

10. Main Roles in Scrum ..... 31

11. A Sample Product Backlog ..... 33

12. A Sample Sprint Backlog ..... 34

13. The Framework of Scrum ..... 36

14. Scrum Lifecycle ..... 37

15. History of Unified Process ..... 39

16. Two Dimensions of RUP ..... 40

17. Utilization of Disciplines in RUP phases ..... 41

18. Differences between Quantitative and Qualitative Method ..... 45

19. Strengths of Qualitative Methods ..... 46

20. Weakness of Qualitative Methods ..... 47

21. Six Different Types of Case Studies ..... 53

22. Differences Between ABC and XYZ Firm ..... 59

23. Type and Number of Interviews Conducted at ABC Firm ..... 61

24. Type and Number of Interviews Conducted at XYZ Firm ..... 62

25. Categories, Concepts, and Data Related to ABC ..... 67

26. C# Coding Standard ..... 85

27. Categories, Concepts, and Data Related to the XYZ Firm ..... 102

28. Human Resource Management Issues ..... 131

29. Structured Development Process Issues ..... 136

30. Environment Issues ..... 142

31. Information Systems and Technology Issues ..... 147

32. Disciplines of Hybrid Model and RUP ..... 165

# LIST OF FIGURES

Figure Page

1. Waterfall model lifecycle ..... 8

2. Original royce waterfall model (source: royce, 1970) ..... 10

3. Project resolutions ..... 11

4. Three legs of empirical process control ..... 27

5. Burndown chart ..... 35

6. Flow of scrum ..... 38

7. Basic types of design for case studies ..... 54

8. Multi-case design used for the research ..... 55

9. Data analysis process ..... 65

10. The theoretical model of a human resource management factor ..... 157

11. The theoretical model of a structured development process factor ..... 159

12. The theoretical model of an environmental factor ..... 160

13. The theoretical model of an information system and information technology factor 162

14. Theoretical model for the success of scrum ..... 163

15. A new hybrid model ..... 164

16. The inception phase of a hybrid model ..... 167

# CHAPTER I

# INTRODUCTION

## *The Problem*

Over the past four decades, many software development methods have been created and utilized in the software industry. Each method has different features and characteristics that distinguish it from other methods; in general, these methods can be classified as either a heavyweight or a lightweight method. The heavyweight methods, also considered traditional methods, usually focus on comprehensive planning, heavy documentation, and big design up-front. In contrast, the lightweight methods, also known as agile methods, concentrate (1) more on individuals and interactions than processes and tools, (2) more on working software than comprehensive documentation, (3) value customer collaboration more than contract negotiation, and (4) focus more on responding to change than following a plan.

The traditional methods are still widely used in the software industry because of their straightforward, methodical, and structured nature; they have proved that they can provide high assurance, stability, and predictability. However, they have a number of key shortcomings, including slow adaptation to constantly changing business requirements and a tendency to be over budget and/or behind schedule, delivering fewer features and functions than specified in the requirements. The need for a complete set of requirements prior to design is also a major challenge for the traditional methods due to vague user specifications.

As a remedy for the shortcomings of the traditional methods, agile software development methods, including Scrum, eXtreme Programming (XP), Crystal, and

Adaptive Software Development (ASD), have been created and evolved by practitioners since the 1990s; they are designed to embrace, rather than reject, high rates of change. These new approaches focus mainly on iterative and incremental development, customer collaboration, and frequent delivery through a light and fast development cycle. Many researchers have reported that agile methods have the potential to provide a higher level of customer satisfaction, lower bug rates, a shorter development cycle, and a quicker adaptation to rapidly changing business requirements.

In spite of the potential benefits of the agile methods, many organizations are reluctant to throw their traditional methods away and jump into agile methods. Their reluctance is the result of several issues, including: (1) the agile methods significantly reduce the amount of documentation and rely heavily on tacit knowledge, (2) these methods have not been sufficiently tested for mission/safety-critical projects, (3) belief that these methods are not adequate for highly stable projects, (4) a concern that agile methods can be successful only with talented individuals who favor many degrees of freedom, and (5) that agile methods are not appropriate for large-scale projects.

Although many positive benefits of the agile methods have been published, there have been few empirical field studies on the negative aspects of various agile methods. The negative aspects of the agile methods mentioned earlier imply that there are issues, problems, and challenges faced in developing high-quality software products using these methods. Identifying the issues, problems, and challenges of the agile methods should be more beneficial to organizations considering them than merely showing their positive benefits. Organizations can learn more lessons from the examining the negative aspects and learning how to avoid the obstacles in adopting the agile methods. Another problem

of many published papers describing positive benefits is that most agile methods have been primarily applied to small-scale and relatively simple projects. It is not clear whether agile methods can provide end users with the desired quality, in a timely manner, on large-scale and mission-critical projects. Therefore, it is worthwhile to conduct a research project to identify the issues and challenges of agile methods and assess the applicability of agile methods to large-scale and mission-critical projects.

For this research, two research sites were chosen for an in-depth case study. Both organizations have produced several high-quality software products through an agile software development methodology with one particular method, called Scrum. The rationale for the selection of the Scrum method among the other available agile methods was (1) Scrum is a widely used agile method in the software industry, in particular in the United States, and (2) the Scrum method claims to be suitable to any size of project. This study also investigated the framework (roles, ceremonies, and artifacts) and the empirical process (visibility, inspection, and adaptation) of the Scrum method in both small- and large-scale projects.

An exploratory research process using observations, surveys, documentation, and interviews was conducted at two organizations. The contribution of this research is five-fold: (1) it identifies critical issues and challenges that may affect the quality of the application of agile methods, (2) it illustrates how agile methods can be adopted and utilized to effectively support the development of small-scale, large-scale, and mission-critical projects, (3) it provides lessons for using Scrum obtained from the field to assist Scrum practitioners, (4) it provides management guidelines to help many organizations avoid and overcome obstacles when adopting the Scrum method as a future software

development method, and (5) it suggests a new framework for further research on the application of traditional and agile methods.

## *Problem Statement*

The few empirical field studies of the negative aspects of agile software development methodologies have failed to identify how the methods can still be useful to organizations and have not assessed their applicability for large-scale and mission-critical projects.

# CHAPTER II

# REVIEW OF LITERATURE

## *Introduction and Purpose of the Review of Literature*

The purpose of this literature review is to examine the existing literature to determine if there are sufficient evidences on issues and challenges in agile software development methods, in particular, in Scrum. This requires a review of the literature related to the evolvement of software development methods. The first section describes the first generation of software development methods and includes a review of the rationale for the appearance of engineering-discipline-based software development methods. The second section reviews the nature of traditional software development methods and illustrates the development lifecycle of the waterfall method. This section of the literature review also includes the shortcomings of the waterfall method and the research results of the Standish Group on projects conducted with traditional software development methods. The third section reviews the Manifesto for Agile Software Development and the principles behind the Agile Manifesto. It includes a review of discriminators between traditional methods and agile methods. It also reviews the agile methods for large-scale projects. This section concludes with a review of a comparison between traditional and agile methods on their characteristics, strengths, and weaknesses. The fourth section reviews the origin of the Scrum method and the empirical process theory that Scrum employs. This section also includes a framework of Scrum and concludes with flow of the Scrum method. The fifth section reviews the Rational Unified Process (RUP), which is used to propose a new framework for future study. This section includes a history, the four phases, and the disciplines of RUP. The last section reviews

quantitative methods and qualitative methods and concludes with how qualitative methods are utilized in Management Information Systems.

### *Early Stage of Software Development Methods*

The early stages of software development can be summarized as “code and fix, code-some-more, fix-some-more” (Fowler, 2005; Leffingwell, 2007). This very simple scheme can be considered as the first generation in the history of software development methods. The fundamental concept of the scheme is to write code first without putting much effort on pre-planning and pre-designing, and to fix bugs later if any are found. This illustrates that the early stage of software development processes did not include any structured and disciplined software development methods. This kind worked very well for small-scale and relatively simple projects. However, as the size of projects increased, developers realized that they spent more time on fixing bugs than writing code. This led to a dramatic decrease in efficiency and predictability of software development. The more software developers worked on large projects, the more they recognized that they needed a methodology with which to impose discipline on the process of software development. The concept of the early structured software development methods were borrowed from the engineering disciplines, which place heavy weight on precise planning (Fowler).

Engineering-discipline-based development methods can be viewed as plan-driven methods, where the documentation of a complete set of requirements precedes architectural and high-level design, development, and implementation (Awad, 2005). The plan-driven software development methods require extensive planning, codified

processes, and rigorous reuse (Boehm, 2002). The plan-driven methods also work best when developers know all of the requirements in advance and when requirements are relatively stable (Hickey & Davis, 2004; Schach, 2004). Due to these factors, these kinds of methods came to be known as heavyweight methods and are also considered traditional software development methods (TSDMs). The next section reviews the nature of the traditional software development method.

### *Traditional Software Development Methods*

Many software development methods that control a software development project have evolved. One well-known TSDM is the Waterfall model, which utilizes a structured and sequential progression between defined phases: planning, analysis, design, implementation, and maintenance. According to Dennis, Wixonm, and Tegarden (2005), the planning phase, which occupies typically about 15% of the total Systems Development Life Cycle (SDLC), is the fundamental process to identify the scope of the new system, understand why a system should be built, and how the project team will go about building it through technical, economical, and organizational feasibility analyses. The analysis phase, about 15% of the SDLC, analyzes the current system, its problems, and then identifies ways to design the new system through requirements gathering. The design phase, 35% of the SDLC, decides how the system will operate in terms of hardware, software, and network infrastructure. The implementation phase occupies about 30% of SDLC and is the phase where actual coding occurs. The maintenance phase occupies the remaining 5% of SDLC and focuses on going-live, training, installation,

support plan, documentation, and debugging. Figure 1 and Table 1 below show a typical waterfall lifecycle and deliverables, respectively.

As we can see in the figure and the table, each phase must be accomplished before the following phase can begin and each phase cannot go back to the previous phase just as water in the waterfall cannot climb up once it reaches a lower position.

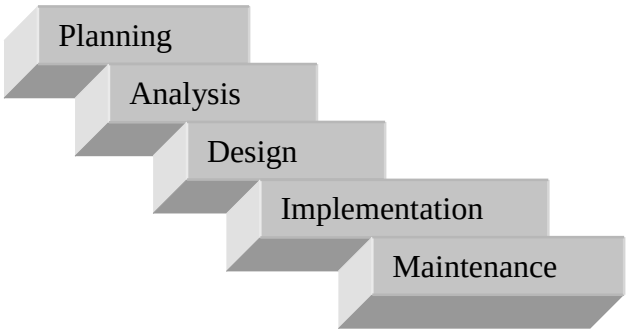



*Figure 1*. Waterfall model lifecycle.

Table 1

*Waterfall Model Deliverables*

| Phases | Deliverables |
|---|---|
| Planning Phase | Planning Specifications |
| Analysis Phase | Analysis Specifications |
| Design Phase | Design Specifications |
| Implementation Phase | Completed Product |

The original waterfall model was formally described by Royce (1970). He believed that managing a large software development should be an iterative process rather than a sequential process. He argued that implementation without having an iterative relationship between development phases would be risky and failure-prone. Interestingly, the original Royce model emphasized heavy feedback and iterations but has been extensively misinterpreted as a fixed sequential process that assumes one can get

things right in a single pass (Leffingwell, 2007). Figure 2 shows the original Royce waterfall model.

Over the past four decades, traditional waterfall-style software development methods have been widely used for large-scale projects in the software industry and in the government sector due to their straightforward, methodical, and structured nature as well as their capability to provide predictability, stability, and high assurance (Boehm & Turner, 2003; Fruhling & De Vreede, 2006). However, TSDMs have a number of key shortcomings, including slow adaptation to constantly changing business requirements, and a tendency to be over budget and behind schedule with fewer features and functions than specified (Boehm, 2002; Boehm & Turner, 2003; Brooks, 1995; Schach, 2004; Sommerville, 2004; Watson, Kelly, Galliers, & Brancheau, 1997). The conventional methods also have failed to provide dramatic improvements in productivity, reliability, and simplicity (Brooks, 1995). Boehm and Philip (1988), and Jones (1997), reported that during their project development experience, requirements often changed by 25% or more. Williams and Cockburn (2003) also mentioned that TSDMs were not initially designed to respond to requirements change occurring in the middle of the development process. They also mentioned that the ability to take appropriate action in response to a change often determines the success or failure of a software product.

One interesting research study conducted by the Standish Group regarding 8,380 projects from 365 respondents representing companies across major industry segments, shows that only a small percentage of projects (16.2%) with traditional methods were completed on-time and on-budget with all features and functions specified. However,

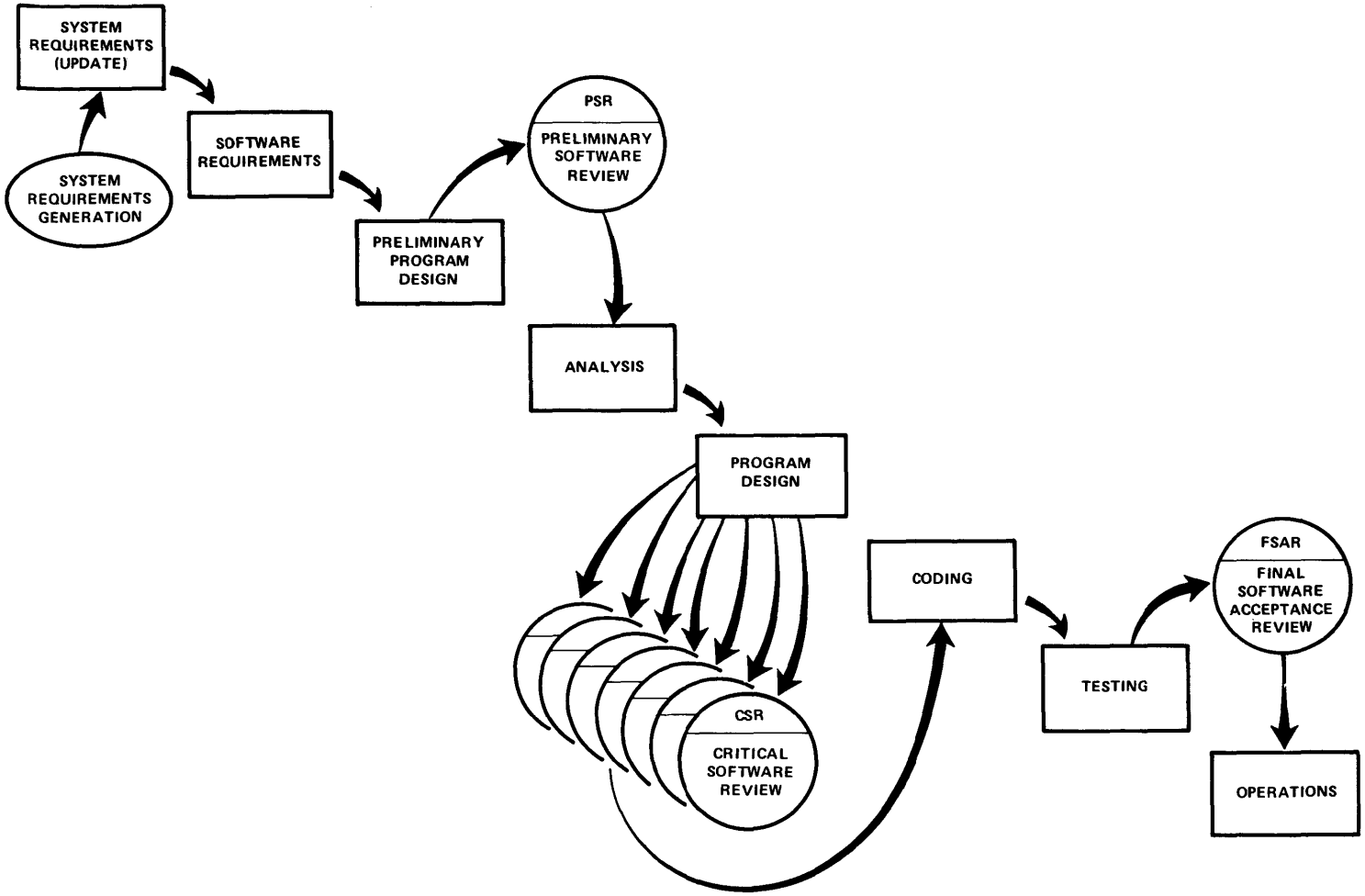



*Figure 2*. Original Royce waterfall model. (Source: Royce, 1970)

52.7% of the projects were completed over-budget, over the time estimate, and/or offering less features and functions; 31.1% of projects were canceled at some point during the development cycle (Standish Group, 1994). Figure 3 displays the result of the research. A more recent study by the same group (Standish Group, 2001) still showed only 28 percent of IT projects were completed on time, on budget and with all the features and functions originally specified. Further, the study found that 23% of the projects failed and 49% of the projects were challenged.

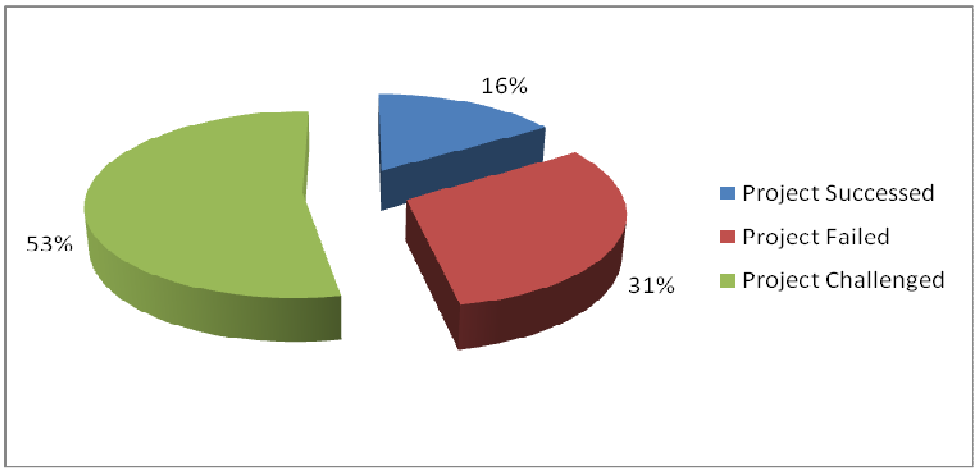



*Figure 3*. Project resolutions.

In another study of 1,027 IT projects in the United Kingdom, Thomas (2001) also reported that scope management related to attempting waterfall practices was the single largest contributing factor for failure. Laffingwell (2007, p. 20) mentioned four key assumptions with the waterfall model that simply turned out to be incorrect. The four assumptions included (1) there exists a reasonably well-defined set of requirements if we only take the time to understand them, (2) during the development process, changes to requirements will be small enough that we can manage them without substantially rethinking or revising our plans, (3) system integration is an appropriate and necessary

process, and we can reasonably predict how it will go based upon architecture and planning, and (4) software innovation and the research and development that is required to created a significant new software application can be done on a predictable schedule.

To address some of the traditional methods' shortcomings, agile methods have been proposed (Highsmith & Cockburn, 2001). The next section explains the characteristics and principles of agile software development methods.

## *Agile Software Development Methods*

### *Overview of Agile Methods*

As a remedy for the traditional software development methods' shortcomings, agile software development methods (ASDMs) were developed. The movement to ASDMs started in the mid-1990s, by many practitioners in parallel, in different languages, different locations, and different project contexts (Leffingwell, 2007). William and Cockburn (2003) mentioned that eXtreme Programming (XP), Scrum, Crystal, and Adaptive Software Development (ASD) were developed in the U.S. by Ken Beck and Eric Gamma, Ken Schwaber and Jeff Sutherland, Alistair Cockburn, and Jim Highsmith, respectively. Dynamic Systems Development Method (DSDM) is a well-documented agile method created by a European consortium of companies and was commercially adopted in Europe (Leffingwell). Feature Driven Development (FDD) was developed in Australia and has contributed to the scaling of agile methods. Table 2 shows country names and founders associated with various agile methods.

The nutshell of ASDMs can be summarized as iterative and incremental development, adaptability throughout the systems development life cycle, minimal

planning, light and fast development cycles, people-centric development, customer collaboration, and frequent delivery. In 2001, seventeen practitioners met at Snowbird, Utah, to discuss if there was anything in common among the various agile methods (Cockburn, 2007), and they created the Manifesto for Agile Software Development (Beck et al., 2001), which revealed what items were considered valuable by ASDMs. As shown in Table 3, ASDMs concentrate (1) more on individuals and interactions than processes and tools, (2) more on working software than comprehensive documentation, (3) value customer collaboration more than contract negotiation, and (4) focus more on responding to change than following a plan.

Table 2

*List of ASDMs*

| **Countries** | **ASDMs** | **Founders** |
|---|---|---|
| U.S.A. | • Extreme Programming (XP) | • Kent Beck, Eric Gamma |
| | • Scrum Method | • Ken Schwaber, Jeff Sutherland |
| | • Crystal Methods | • Alistair Cockburn |
| | • Adaptive Software Development (ASD) | • Jim Highsmith |
| | • Lean Software Development | • Tom and Mary Poppendieck |
| Europe | • Dynamic Systems Development Method (DSDM) | • Dane Faulkner |
| Australia | • Feature Driven Development (FDD) | • Peter Code, Jeff DeLuca |

Table 3

*Manifesto for Agile Software Development*

| More Valuable Items | Less Valuable Items |
|---|---|
| Individuals and interactions | Processes and tools |
| Working software | Comprehesive documentation |
| Customer collaboration | Contract negotiation |
| Responding to change | Following a plan |

Table 4

*Principles Behind the Agile Manifesto (Source: Beck et al., 2001)*

| No. | Principles |
|---|---|
| 1 | Our highest priority is to satisfy the customer through early and continuous delivery of valuable software. |
| 2 | Welcome changing requiremts, even late in development. Agile processes harness change for the customer's competitive advantage. |
| 3 | Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale. |
| 4 | Business people and devlopers must work together daily throught the project. |
| 5 | Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done. |
| 6 | The most efficient and effective method of conveying informaiton to and within a development team is face-to-face conversation. |
| 7 | Working software is the primary measure of progress. |
| 8 | Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely. |
| 9 | Continuous attention to technical excellence and good design enhances agility. |
| 10 | Simplicity—the art of maximizing the amount of work not done—is essential. |
| 11 | The best architectures, requirements and designs emerge from self-organizing teams. |
| 12 | At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly. |

The twelve principles behind the agile manifesto also presented the characteristics of ASDMs (Beck et al., 2001). As shown in Table 4, ASDMs (1) satisfy the customer through early and continuous delivery of software, (2) embrace changing requirements,

even in late in the development cycle, (3) deliver working software frequently, (4) work daily with business people, (5) facilitate motivated people, provide them with a good environment and support, and trust them, (6) assist face-to-face conversation within a development team, (7) use working software as a primary measure of progress, (8) promote sustainable development and keep sponsors, developers, and users moving at a constant pace, (9) pay attention to technical excellence and good design, (10) maintain simplicity, (11) promote self-organizing teams, and 12) foster inspections and adaptations. The Agile Manifesto and its supporting twelve principles provide the basic philosophy of the agile methods, and all applied agile best practices can be directly correlated to them (Leffingwell, 2007).

There are many different characteristics between ASDMs and TSDMs. Boehm (2002), for example, reports nine agile and heavyweight discriminators as shown in Table 5. He believes the primary objective of using ASDMs is rapid value, whereas the primary objective of TSDMs is on high assurance. He also believes that ASDMs are a better choice when requirements are either not known at the beginning of the project, are largely emergent as the project progresses, or change rapidly, whereas TSDMs are better when requirements are known at the early stage of the project and largely stable throughout the duration of the project. Regarding the involvement of customers, Boehm thinks that ASDMs require dedicated, knowledgeable, and collaborated customers, whereas TSDMs do not need co-located onsite customers; rather, they focus more on contract provisions. In ASDMs, developers should be agile, knowledgeable, co-located, and collaborative. In TSDMs, developers should be plan-oriented and have adequate skills to access external knowledge. One more very noticeable difference is that re-

factoring in ASDMs is cheaper than TSDMs. ASDMs have unknown risks, which can have a major impact on a project, whereas TSDMs have well-understood risks and their impact on a project is known. Overall, as shown in the table, Boehm believes ASDMs should be used for small teams and projects. If the size of the team and projects are large, he suggests TSDMs.

Table 5

*Discriminator Between ASDMs and TSDMs (Source: Boehm, 2002)*

| Project Characteristics | Agile Discriminator | Heavyweight Discriminator |
|---|---|---|
| Primary objective | Rapid Value | High Assurance |
| Requirements | Largely emergent, rapid change, unknown | Knowable early, largely stable |
| Size | Smaller teams and projects | Larger teams and projects |
| Architecture | Designed for current requirements | Designed for current and foreseeable requirements |
| Planning and Control | Internalized plans, qualitative control | Documented plans, quantitative control |
| Customers | Dedicated, knowledgeable, collaborated, collocated onsite customers | As needed customer interactions, focused on contract provisions |
| Developers | Agile, knowledgeable, collocated, and collaborative | Plan-oriented; adequate skills access to external knowledge |
| Refactoring | Inexpensive | Expensive |
| Risks | Unknown risks, Major Impact | Well understood risks, Minor impact |

The next section explains lean manufacturing principles which have a long history of generating dramatic improvements in the field of manufacturing (Poppendieck & Poppendieck, 2003) and describes how these principles were accepted into agile methods.

*Lean Manufacturing Principles and Agile Methods*

In addition to the nine discriminators listed earlier, ASDMs have several other distinct attributes. These attributes come mainly from the basic principles of lean industrial practices. As shown in Table 6, Poppendieck and Poppendieck (2003) identified seven principles of lean manufacturing and explained how these principles could be directly applied to software development. The seven disciplines include (1) eliminate waste, (2) amplifying learning, (3) decide as late as possible, (4) deliver as fast as possible, (5) empower the team, (6) build integrity, and (7) see the whole.

Some of these principles are actually embedded into ASDMs. The first significant attribute of agile methods adopted from the lean principles is the elimination of waste. This concept was first introduced by Taiichi Ohno, considered the father of the Toyota Production System. This approach considers any activities that do not improve the quality of a final product as waste. Shigeo Shingo (1981), one of the masterminds of the Toyota Production System, listed seven types of manufacturing waste as shown in the left side of Table 7, the right side represents the corresponding seven wastes of software development. In following this principle, ASDMs avoid preparing heavy documents, models, and diagrams. Instead, ASDMs suggest writing effective guidelines and a set of rules.

Table 6

*Principles of Lean Manufacturing*

| No. | Principle | Main concept |
|---|---|---|
| 1 | Eliminate waste | • Remove any activities that do not improve the quality of a final product.<br>• Avoid heavy documents, models, and diagrams.<br>• Write effective guidelines and a set of rules. |

| No. | Principle | Main concept |
|---|---|---|
| 2 | Amplifying learning | • Create software through iterative and incremental method.<br>• Gather productive feedbacks from each iteration and apply them to the next iteration. |
| 3 | Decide as late as possible | • Any process should provide a capacity for change by delaying a decision as late as possible. |
| 4 | Deliver as fast as possible | • Deliver small but working subsystems as fast as possible.<br>• Frequent delivery of working versions of a final system is a key to rapid development.<br>• Early and frequent delivery of small software components increases the potential of success. |
| 5 | Empower the team | • Team is responsible for success or failure of the project.<br>• Team should be autonomous in organization and management of projects. |
| 6 | Build integrity in | • A system should work smoothly and cohesively |
| 7 | See the whole | • Experts should be able to see a whole system<br>• Overall performance should have more weight than a local maximization. |

Table 7

*The Seven Wastes (Source: Poppendieck & Poppendieck, 2003)*

| No. | The Seven Wastes of manufacturing | The Seven Wastes of Software Development |
|---|---|---|
| 1 | Inventory | Partially Done Work |
| 2 | Extra Processing | Extra Processes |
| 3 | Overproduction | Extra Features |
| 4 | Transportation | Task Switching |
| 5 | Waiting | Waiting |
| 6 | Motion | Motion |
| 7 | Defects | Defects |

The second significant attribute of agile methods is an adaptive development process, which draws on the two lean principles of “amplifying learning” and “decide as late as possible.” The lean principle “amplifying learning” is based on the concept that

"Development is an exercise in discovery while production is an exercise in reducing variation, and for this reason, a lean approach to development results in practices that are quite different than lean production practices." (Poppendieck & Poppendieck, 2003, p. xxv). The lean principle "decide as late as possible" provides a capacity for change by delaying decisions as late as possible. ASDMs follow with these principles by emphasizing adaptive software development, which requires iterative and incremental development through productive feedback. Satzinger, Jackson, and Burd (2005) mentioned that some projects were reasonably predictable and could be managed sequentially but most projects are less predictable, demanding an iterative and adaptive approach to development.

The third significant attribute of agile methods is rapid software development, influenced by the lean principle "deliver as fast as possible." The concept is based on the fact that customers like rapid delivery, and rapid delivery can provide customers with flexibility. ASDMs try to deliver small working subsystems as fast as possible rather than waiting to show the customer the final product until it is complete. Fowler (2005) mentioned that frequent delivery of working versions of a final system was a key to rapid development. These working systems represent only a portion of the whole system, but should be good enough to get customer feedback. The study of 23,000 projects conducted by Standish Group International (1994) revealed that early and frequent delivery of small software components increases the potential for success.

The fourth significant attribute of agile methods is an autonomous and self-organizing team, which mirrors the lean principle "empower the team." This concept is based on the assumptions that (1) a mature organization looks at the whole system and

not on optimizing disaggregated parts, and (2) a mature organization focuses on learning effectively and empowers the people who do the work to make decisions. ASDMs focus more on team work than TSDMs. As a whole, the team is responsible for the success or failure of the project. Teams should be autonomous in term of organization and management of projects.

The last two lean manufacturing principles in the table did not directly affect the attributes of the agile methods but these principles are somewhat embedded into agile practices. Regarding the principle of "build integrity in," Brooks (1995, p. 255) stated "Conceptual integrity means that the system's central concepts work together as a smooth, cohesive whole, and it is a critical factor in creating perceived integrity." The "see the whole" principle puts more emphasis on maximizing over all system performance than maximizing the performance any particular part of the system.

### *Characteristics, Strengths, Weaknesses of TSDMs and ASDMs*

The ASDMs have the potential to provide higher customer satisfaction, lower bug rate, shorter development cycles, and quicker adaptation to rapidly changing business requirements (Boehm, 2002; Boehm & Turner, 2003; Parnas, 2006). Parrish (2004) argues that ASDMs provide increased quality, shorter time to market, better efficiency, and greater customer satisfaction. Miller and Larson (2005) also believe that ASDMs emphasize close collaboration between the users and developers of a project, and relatively quick development cycles that can react to changing requirements. An Australian group, Shine Technologies (2003) surveyed 131 respondents of teams and companies that had employed agile methods and found that (1) 93% stated that

productivity was better or significantly better, (2) 49% stated that costs were reduced or significantly reduced, (3) 88% stated that quality was better or significantly better, and (4) 83% stated that business satisfaction was better or significantly better. Beck (2000) reported that in the original XP project at Daimler Chrysler, it took 12 to 15 people 2 years to write and deploy a system that a team of 30 had failed to deliver in the prior 4 years. Spencer (2005) also stated that "our implementation of agile practices helps us find bugs earlier, helps us achieve higher quality, and helps us work well with QA."

According to a report by Forrester Research (2005), only 14% of North American and European enterprises use agile software development processes, and another 19% are either interested in adopting agile methods or already planning to do so.

So far, we have discussed various strengths, weaknesses, and characteristics of both TSDMs and ASDMs. These were summarized in Table 8 and the next section reviews the literature on agile methods for large-scale projects.

*Agile Methods for Large-Scale Projects*

Most agile methods have primarily been applied to small to medium size projects such as internet and web-based information systems. It is not clear if agile methods are used on large-scale projects that they can provide end-users with the desired quality in a timely manner (Marrington, Hogan, & Thomas, 2005). However, some researchers have reported that large-scale and complex projects have benefited from suitably tailored agile development methods (Bowers, May, Melander, Baarman, & Ayoob, 2002; Lippert et al., 2003; Cao, Mohan, Xu, & Ramesh, 2004; Lindvall et al., 2004). Bowers et al. (2002) examined whether the XP method can handle large-scale and life-critical software

systems. The authors adopted the XP method to redesign their public safety communication systems, which consists of over a million lines of C language code.

Table 8

*Characteristics, Strengths, and Weaknesses of TSDMs and ASDMs*

| | TSDMs | ASDMs |
|---|---|---|
| Characteristics | • Extensive planning<br>• Codified process<br>• Rigorous reuse<br>• Heavy documentation<br>• Big design up front | • Iterative and incremental<br>• Customer collaboration<br>• Frequent delivery<br>• People centric<br>• Light and fast development cycle |
| Strengths | • Straightforward, methodical, and structured nature<br>• Predictability, stability, and high assurance | • Short development cycle<br>• High customer satisfaction<br>• Low bug rate<br>• Quick adaptation to rapidly changing business requirements |
| Weaknesses | • A slow adaptation to rapidly changing business requirements<br>• A tendency to be over budget<br>• A tendency to be behind schedule<br>• Difficult to create a complete set of requirements up front | • Significant document reduction and heavy dependent on tacit knowledge<br>• Not sufficient test for mission/safety-critical projects<br>• Not adequate for highly stable projects<br>• Can be successful only with talented individuals who favor many degrees of freedom<br>• Not appropriate for large-scale projects |

They used the XP method to mitigate risks with early, frequent feedback. However, they did not use every part of the XP method. Instead, they adopted some practices, dropped others and supplemented others with practices from other fields. This paper revealed the possibilities for applying the XP method to large-scale and life-critical projects if the XP method was modified to fit into the specific application development environment. Lippert et al. (2003) also examined whether the XP method was appropriate for large and long term projects. They indicated that a suitably adapted agile development process (in particular XP) was ideal for long-term projects and the development of large systems. This is contradictory to the preferences of many information technology (IT) managers who often consider XP as a slightly chaotic methodology. Lippert et al. mentioned that they followed the recommended practice of adapting XP to their specific project. They also developed methodological extensions to XP for use in a number of areas in which questions and problems frequently occur. The majority of studies on large-scale projects have been conducted using the XP method, which was initially designed for small-scale projects with less than 10 developers and a product that would not be excessively complex (Beck, 2000).

Among the various agile methods, the Scrum method was selected for this study because Scrum is one of the most widely adopted agile methods in the U.S. software industry (Leffingwell, 2007; Williams & Cockburn, 2003), and Scrum is suitable for any size of projects (Schwaber & Beedle, 2002). The next section explains the roots, a history, the empirical process control, and the framework of the Scrum method.

## *Scrum Software Development Method*

### *The Philosophical Roots of Scrum*

The Scrum software development method is an agile process that can be used to manage and control complex software and product development using iterative and incremental practices (Advanced Development Methods, 2009; Schwaber, 2004, 2007, 2008; Schwaber & Beedle, 2002) and is an enhancement of the iterative and incremental approach to delivering objected-oriented software (Schwaber, 1996). Leffingwell (2007, p. 41) also defined the Scrum method as ". . . a lightweight agile project management method based on small, empowered, self-organizing teams; complete visibility; and rapid adaptation." The origin of term scrum came from the popular sport rugby, in which fifteen players on each team compete against each other. While the term scrum refers to the strategy used for getting an out-of-play ball back into play in rugby, it was first used to describe hyper-productive development processes in Japan (Takeuchi and Nonaka, 1986). Three strategies from rugby, including a holistic team approach, constant interaction among team members, and unchanging core team members, are adopted into Scrum's management and control processes. Takeuchi and Nonaka (1986, p. 137) noted:

> This new emphasis on speed and flexibility calls for a different approach for managing new product development. The traditional sequential or 'relay race' approach to product development - exemplified by the National Aeronautics and Space Administration's phased program planning (PPP) system - may conflict with the goals of maximum speed and flexibility. Instead, a holistic or 'rugby' approach - where a team tries to go the distance as a unit, passing the hall back and forth - may better serve today's competitive requirements.

Takeuchi and Nonaka also described six principles that contribute substantially to the Scrum philosophy. Each principle correlates directly to many of the principles of the Agile Manifesto that we previously discussed. Their six principles are (1) built-in

instability, (2) self-organizing project teams, (3) overlapping development phases, (4) multi-learning, (5) subtle control, and (6) organizational transfer of learning. Table 9 shows these principles with explanations.

Table 9

*Six Principles of the New Product Development Process*

*(Source: Leffingwell, 2007, p. 43)*

| No. | Principles | Contents |
|---|---|---|
| 1 | Built-in instability | A principle philosophy for management is to provide a vision and challenge to the team but not to provide the specific steps for how the team is to accomplish these objectives. |
| 2 | Self-organizing project teams | Self-organizing teams exhibit three conditions: autonomy, self-transcendence, and cross-fertilization. |
| 3 | Overlapping development phases | In Scrum, the lines between product definition, design, code, and test are blurred. Product definition derives design, which affects product definition |
| 4 | Multi-learning | By organizing in an environment that offers multiple learning opportunities, teams are in constant and close contact with each other as well as with outside sources of information via the customer proxy or direct involvement of the team with the customer. |
| 5 | Subtle Control | Scrum provides daily and monthly objective checkpoints for each project. Management provides additional control by creating a proper and open work environment, encouraging the team to interact with customers, and establishing reward systems based on team, rather than individual behavior. |
| 6 | Organizational transfer of learning | Scrum teams in particular, and agile teams in general, routinely exhibit transfer of learning outside their project team. Learning may be driven by the excitement generated by more effective processes and self-empowered teams or by the objective results, but in any case, good news spreads quickly. |

*The History and Practices of Scrum*

The Scrum process was developed by Schwaber and Sutherland (Schwaber & Beedle, 2002). The former developed and formalized the Scrum process for system

development while he was at his company, Advanced Development Methods (ADM), in the early 1990s. The latter developed many of the initial thoughts and practices for Scrum when he was at Easel Corporation as a vice president of Object Technology in 1994. By a joint effort of both, the Scrum process was first introduced to the public at the conference of Object-Oriented Programming, Systems, Languages and Applications (OOPSLA) in 1996 (Schwaber, 1996). The Scrum process looks simple, but is practical enough to deeply influence the work experience and to capture key agile characteristics (Larman, 2007). Some key practices of Scrum (Laffingwell, 2007, p. 44; Larman, p. 109) include the following point. (1) Self-managing, cross-functional, self-directed, self-organizing, and collocated teams of eight or fewer team members develop software in Sprints. (2) Sprints are iterations of fixed 30-day duration, where each sprint delivers incremental, tested functionality of value to the user. (3) Work within a Sprint is fixed. Once the scope of a Sprint is chosen, no external addition of work can be added except by the development team. (4) The Scrum master mentors and manages the teams that are responsible for delivery of successful outcomes at each sprint. (5) All work to be done is carried as a Product Backlog, which includes requirements to be delivered, the defect workload, as well as infrastructure and design activities. (6) The Product Backlog is developed, managed, and prioritized by the product owner, who is an integral member of the team and who has the primary responsibility of interfacing with the external customers. (7) A daily stand-up meeting with special questions is a primary communication method. (8) Scrum focuses heavily on time-boxing. Sprints, stand-up meetings, release review meetings are all completed in prescribed times. (9) Scrum allows requirements, architecture, and design to emerge over the course of the project.

(10) Scrum provides a demo to external stakeholders at end of each iteration, and each iteration is client-driven adaptive planning.

The Scrum method consists of three main elements - the empirical process control (visibility, inspection, and adaptation), the framework (roles, ceremonies, and artifacts), and the workflow. The following sections describe in detail these three main elements.

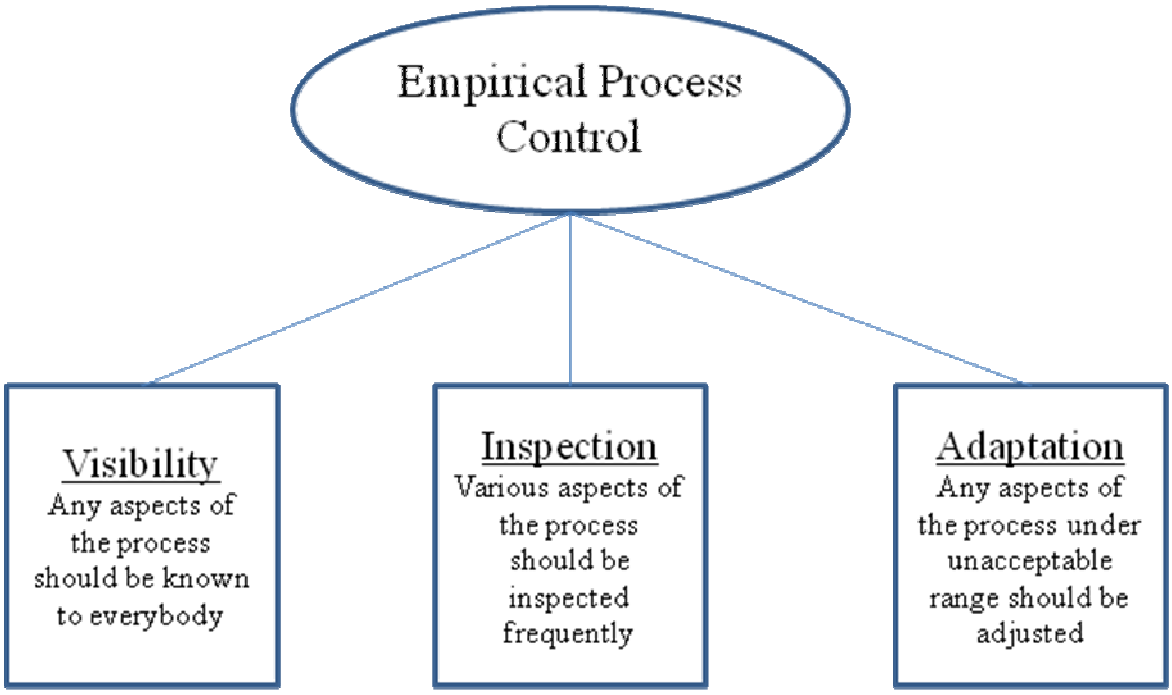



*Figure 4*. Three legs of empirical process control.

*Empirical Process Control*

The co-founder of the Scrum process, Schwaber (2004, 2008), has argued that the Scrum process employs an empirical process control which has three legs underlying all of its implementations: Transparency (Visibility), Inspection, and Adaptation. Transparency or visibility means that any aspects of the process that affect the outcome must be visible and known to everybody involved in the project process. Inspection requires that various aspects of the process be inspected frequently enough so that

unacceptable variances in the process can be detected. Adaptation requires that the inspector should adjust the process if one or more aspects of the process are in an unacceptable range. Figure 4 visualizes the empirical process.

Schwaber (2002, p. 24) introduced a noteworthy story, cited below, that points out that the system development process is complex and unpredictable, hence it should be managed by a process control model that is empirical.

> I wanted to understand why my customers' methodologies didn't work for my company, so I brought several methodologies to process theory experts at the DuPont Experimental Station in 1995. These experts, led by Babatunde Ogannaike, are the most highly respected theorists in industrial process control. They inspected the system development processes that I brought them. I have rarely provided a group with so much laughter. They were amazed and appalled that my industry, system development, was trying to do its work using a completely inappropriate process control model. They said systems development had so much complexity and unpredictability that it had to be managed by a process control model they referred to as "empirical". They said this was nothing new, and all complex processes that weren't completely understood required the empirical model.

Schwaber (2004) mentioned that a code review can be analyzed with the empirical process control model described above. Any code written by developers should be visible to everybody (transparency). The most experienced and knowledgeable developers can review the code (inspection). If there is room to improve the code, reviewers' comments and suggestions should be reflected in the code (adaptation).

*Framework of Scrum*

The framework of Scrum consists of three components including roles, ceremonies, and artifacts (Schwaber, 2004). There are three distinct roles in the Scrum process: the Product Owner, the Team and the Scrum master.

*Three Roles in Scrum*. *The Product Owner* is responsible for getting initial and on-going funding for the project by creating the project's overall requirements, return on investment (ROI) objectives, and release plan (Schwaber, 2004). The Product Owner is also responsible for managing and controlling the Product Backlog, which is described in the Scrum artifacts section. The Product Owner should be one person rather than a committee. If any of items in the Product Backlog need to be changed or re-prioritized, this should be done through the Product Owner. The reason for establishing a single Product Owner is to avoid having multiple conflicting lists. The Product Owner should make the Product Backlog visible to everyone so that everyone knows which item has the highest priority. To finish the project successfully, everyone in the organization should follow the decision made by the Product Owner. The Product Owner can be the product manager for commercial development, or the project manager or the user department manager in-house development.

*The Team* is responsible for implementing the functionality described in the requirements. Teams should be self-managing, self-organizing, and cross-functional to maximize team performance. All of the team members are responsible for both the success and the failure of sub-systems and the entire system (Schwaber, 2004). Team members decide what items should be accomplished over the next Sprint in the Sprint planning meeting. The team should be autonomous so that it has the power to make a decision, do whatever it needs to do, and ask for any impediments to be removed. Although the team can make a decision on how to do its work, the team's decision should be conform to any existing organization's charters, standard, conventions, architectures,

and technology. It is desirable to create a team with people who have different kinds of skills necessary to meet the Sprint goal.

If the size of the team is larger than eight people, Schwaber and Beedle (2002) strongly recommend breaking them into multiple teams to minimize the interaction and dependencies between team members. They also believe that large teams generate too much complexity for an empirical process. Team members are called pigs because they, like the pigs in the following joke, are committed to the project. Other than team members, everyone else is a chicken. Chickens can attend the Daily Scrum Meeting but should be remain silent. Chickens cannot interfere with the meetings in any way.

> A chicken and a pig are together when the chicken says, "Let's start a restaurant!" The pig thinks it over and says, "What would we call this restaurant?" The chicken says, "Ham n' Eggs!" The pig says, "No, thanks. I'd be committed, but you'd only be involved (Schwaber & Beedle, 2002, p. 42).

The *Scrum Master (SM)* is a new management role introduced by Scrum. According to Schwaber and Beedle (2002), the SM has various duties and responsibilities. The SM is responsible for ensuring that Scrum values, practices, and rules are enacted and enforced. The SM is the driving force behind all of the Scrum practices. The SM sets them up and makes sure they happen. The SM represents management and the team to each other. At the Daily Scrum, the SM listens closely to what each team member reports. The SM compares what progress has been made to what progress was expected, based on Sprint goals and predictions made during the previous

Daily Scrum. SM also tries to remove any impediments imposed on developers (Schwaber & Beedle, 2002, p. 31). Table 10 shows the roles in Scrum.

Table 10

*Main Roles in Scrum*

| Roles | |
|---|---|
| The Product Owner<br><br>• creates the project's overall requirements, return on investment (ROI) objectives, and release plan<br>• manages and controls the Product Backlog<br>• can be the product manager, project manager, or user development manager | The Team<br><br>• is responsible for implementing the functionality<br>• is responsible for both the success and the failure of systems<br>• should be self-managing, self-organizing, and cross-functional<br>• should be committed to the project |
| The Scrum Master<br><br>• is a new management role introduced by Scrum<br>• is responsible for ensuring that Scrum values, practices, and rules are enacted and enforced<br>• is the driving force behind all of the Scrum practices<br>• represents management and the team to each other<br>• tries to remove any impediments imposed on developers | Everyone Else<br><br>• is chicken<br>• cannot interfere with the meetings in any way |

*Ceremonies in Scrum.* There are several ceremonies in the Scrum process including the Daily Scrum Meeting, the Daily Scrum of Scrums Meeting, the Sprint Review Meeting and the Sprint Planning Meeting. The *Daily Scrum Meeting (DSM)* is a 15-minute status meeting to talk about what has been accomplished since the last meeting, what items will be done before the next meeting, and what obstacles developers

have. DSMs facilitate communications, identify and remove impediments to development, highlight and promote quick decision-making, and improve transparency (visibility) as explained in the previous section. The *Daily Scrum of Scrums Meeting (DSSM)* is another short daily meeting and follows the same format as a regular DSM. The main reason for having a DSSM is to synchronize the work between multiple Scrum teams.

The *Sprint Planning Meeting (SPM)* is a monthly meeting, where the Product Owner and Team get together to discuss what will be done for the next Sprint which lasts usually for 30 days. In a SPM, team members break a project into a set of small and manageable tasks so that all the tasks can be completed in one Sprint. The *Sprint Review Meeting (SRM)* is another monthly meeting which is held at the end of the Sprint. An SRM is usually a four-hour time-boxed meeting, where team members present what was developed during the Sprint to the Product Owner and stakeholders.

*Three Artifacts in Scrum.* In addition to the Scrum roles and ceremonies, the Scrum process provides three artifacts, namely the Product Backlog, the Sprint Backlog, and the Burndown Chart. The *Product Backlog* is a collection of functional and non-functional requirements, which are prioritized in order of importance to the business. The items in the Product Backlog are created and maintained by the Product Owner. A simple version of Product Backlog is shown in Table 11. A complete Product Backlog can be found in Appendix A. As shown in Table 11, the requirement column represents various projects needing to be accomplished in the Sprint. The Num column places an internal number for each project, which is used for referential purposes. Some of examples for the

Category column are feature, enhance, defect, and technology. The status column represents “not started,” “underway,” or “complete.” The priority column ranges from one to five and the smaller number represents the higher priority. The estimate column represents the estimated hours needed to finish the project.

Table 11

*A Sample Product Backlog*

| 1 | **Product Backlog** | | | | | |
|---|---|---|---|---|---|---|
| 2 | | | | | | |
| 3 | Requirement | Num | Category | Status | Priority | Estimate |
| 4 | Create login screen | 12 | Enhance | Not started | 5 | 100 |
| 5 | PDA sale capture | 117 | Enhance | Not started | 5 | 300 |
| 6 | Auto-size column | 23 | Enhance | Complete | 2 | 120 |
| 7 | Create JLBBK interface | 120 | Feature | Underway | 1 | 130 |
| 8 | Create log entry | 121 | Feature | Not started | 3 | 500 |
| 9 | Credit card payment | 26 | Defect | Complete | 2 | 200 |
| 10 | Commission calculation | 221 | Feature | Underway | 4 | 700 |
| 11 | Create intake screen | 124 | Feature | Not started | 3 | 800 |

*The Sprint Backlog* is created by team members from the Product Backlog in a way that the high priority items in the Product Backlog are first selected and broken into a set of smaller tasks. When the Product Backlog items are divided into small tasks, team members estimate the completion time for each task. Team members try to make tasks as small as possible so that every task can be accomplished within three days. The Sprint Backlog consists of these small tasks. A simple version of Sprint Backlog is shown in Table 12. A complete Sprint Backlog can be found in Appendix B. As shown in Table

12, the Task Description column represents a set of small tasks that needs to be completed within one cycle of the Sprint.

Table 12

*A Sample Sprint Backlog*

| 1 | **Sprint Backlog** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 2 | | | | | | | | | |
| 3 | Task Description | Originator | Responsible | Status | Hours of work remaining | | | | |
| | | | | | 488 | 344 | 321 | 298 | 255 |
| 4 | Add search functionality | JS | JJ/SS | In progress | 20 | 11 | 11 | 6 | 2 |
| 5 | Create inmate release | JC | AN | In progress | 24 | 16 | 16 | 16 | 8 |
| 6 | Update parameter screen | JC | BB | Not started | 23 | 23 | 23 | 23 | 23 |
| 7 | Release screen enhancement | JS | PB | Complete | 24 | 17 | 10 | 6 | 0 |
| 8 | Nested control (Moved to next Sprint) | JS | SN | Removed | 20 | 20 | 20 | 20 | 20 |
| 9 | Add incident tab in inmate screen | JC | SJ | In progress | 25 | 20 | 16 | 8 | 8 |
| 10 | Format the tab in inmate screen | JC | AN | In progress | 24 | 24 | 16 | 8 | 8 |
| 11 | Disable medication tab in inmate screen | JC | TK | Complete | 23 | 23 | 16 | 8 | 0 |

These lists are created by team members from the Requirement listed in the Product Backlog. The Originator column represents a name of person or people who originate the task. The Responsible column shows a name of person or people who are responsible for. The Status column shows several different status of each task including “Not started,” “In progress,” “Complete,” and “Removed.” The last column shows the hours of work remaining. At the end of the Sprint session the remaining hours for each

task should be zero. As shown in row #8, some of tasks in the Sprint Backlog cannot be completed and are carried over to the next Sprint.

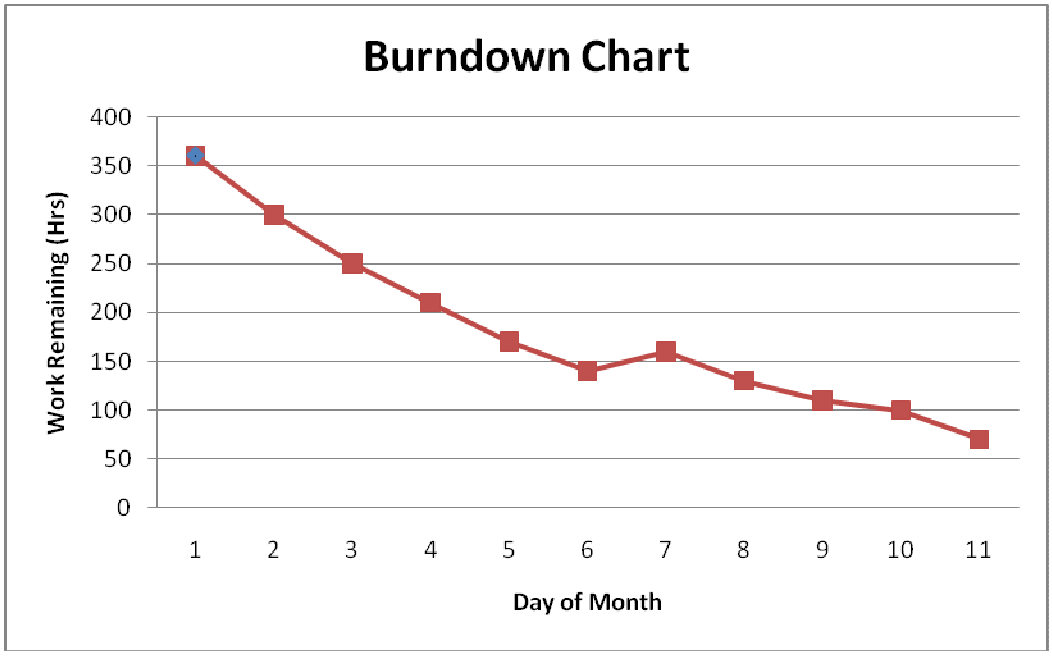



*Figure 5*. Burndown chart.

*The Burndown Chart* is a graphical presentation where work remaining is tracked on the vertical axis and the time periods tracked on the horizontal axis. The chart should be accessible by every member participating in the project. It is a good tool for providing visibility (transparency) to the people who are involved in the project. In Scrum, the Burndown Chart is considered the most critical project data to track (Larman, 2007). It is recommended to post an updated version of the chart each day by the Scrum meeting so that team members can see the current status of each task. An individual and team chart can be used to show individual performance and team velocity respectively. Figure 5 shows a simple version of an individual Burndown Chart. A complete individual and team Burndown chart can be found in Appendix C.

As mentioned at the beginning of this section, the framework of Scrum consists of three components: Roles, Ceremonies, and Artifacts. Table 13 summarizes the three components and major elements of each component.

Table 13

*The Framework of Scrum*

| Roles | Ceremonies | Artifacts |
|---|---|---|
| • The Product Owner<br>• The Scrum Team<br>• The Scrum Master | • Daily Scrum Meeting<br>• Daily Scrum of Scrums Meeting<br>• Sprint Review Meeting<br>• Sprint Planning Meeting | • Product Backlog<br>• Sprint Backlog<br>• Burndown Chart |

*Flow of Scrum*

The Scrum process begins with a vision of the system and a simple plan on Return on Investment (ROI) and release milestones. The vision is described in business terms rather than technical terms. The vision may be unclear at first, but will become more precise as the project moves forward. As mentioned earlier, the Product Owner is responsible for getting initial funding, delivering the vision while maximizing ROI, and creating the Product Backlog. The prioritized items in the Product Backlog are divided into smaller tasks through the Sprint Planning Meeting and placed in the Sprint Backlog. In the Sprint Planning Meeting, the Product Owner explains the content, purpose, meaning, and intentions of each item in the Product Backlog. Team members can ask questions if they do not understand any items in the Product Backlog. All the tasks in the Sprint Backlog are undertaken in Sprints, which are done iteratively until the tasks are

completed. Daily Scrum Meetings are used to review task progress during each Sprint. Figure 6 illustrates the flow of the Scrum process and Table 14 shows a Scrum lifecycle.

Table 14

*Scrum Lifecycle*

*(Source: Larman, 2007, 113)*

| | Pre-Game | | Development | Release |
|---|---|---|---|---|
| | Planning | Staging | | |
| Purpose | Establish the vision, set expectations, and secure funding | Identify more requirements and prioritize enough for first iteration | Implement a system ready for release in a series of 30-day iteration (Sprints) | Operational deployment |
| Activities | Write vision, budget, initial product backlog and estimate items.<br><br>Exploratory design and prototype | Planning<br><br>Exploratory design and prototypes | Sprint planning meeting each iteration, defining the Sprint Backlog and estimates<br><br>Daily Scrum meetings<br><br>Sprint Review | Documentation<br><br>Training<br><br>Marketing & Sales |

*Rational Unified Process*

The Unified Process (UP) is a well-defined, object-oriented system development process originally offered by IBM Rational Software and was developed by Booch, Rumbaugh, and Jacobson (Satzinger et al., 2005).

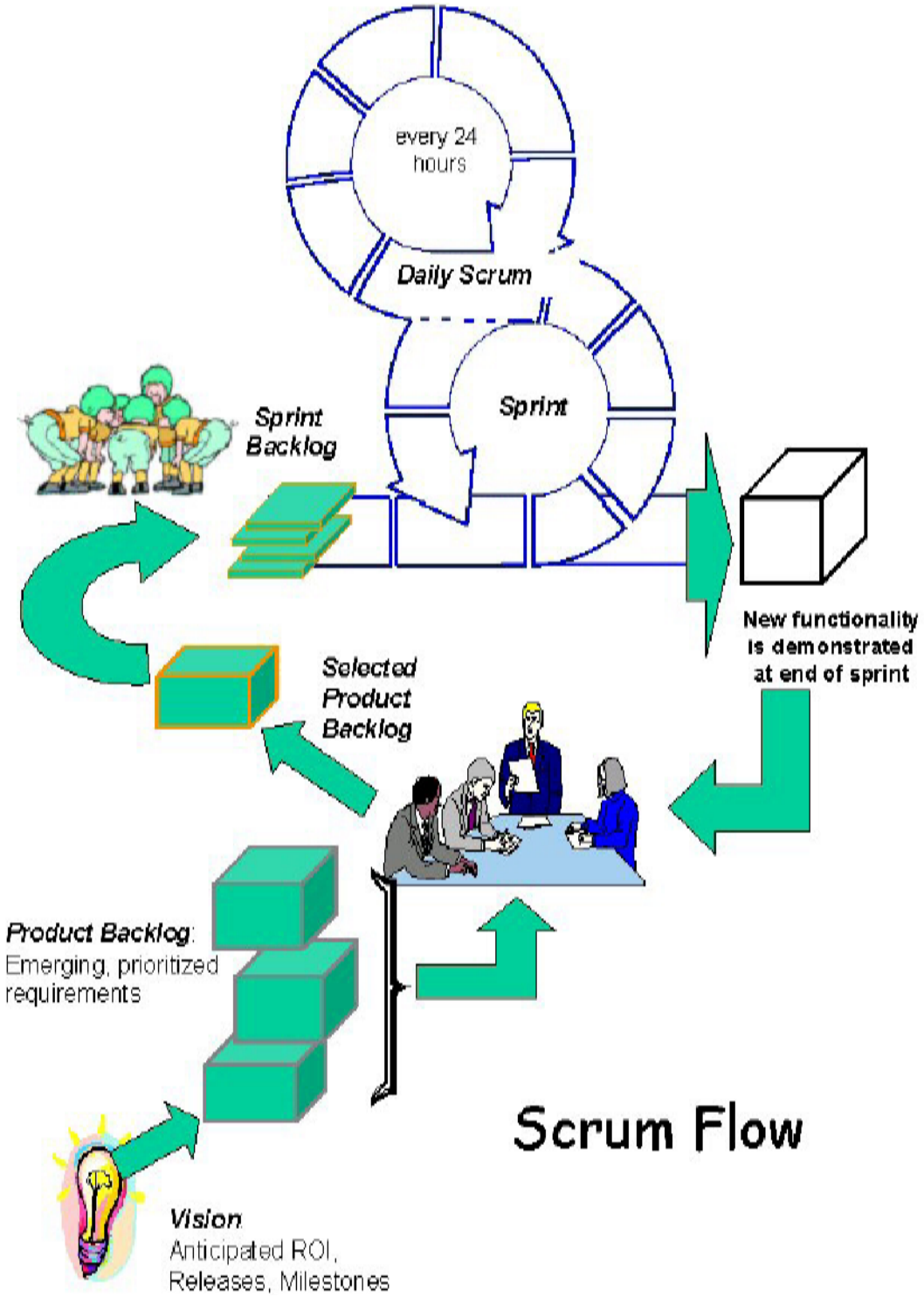



*Figure 6*. Flow of Scrum.
(Source: Hodgetts, 2009)

The UP takes an iterative, requirements-driven, and architecture-centric approach, based on sound engineering principles (Kruchten, 2004). Some of the universal principles of UP includes: (1) adapt the process, (2) balance stakeholder priorities, (3) collaborate across teams, (4) demonstrate value iteratively, (5) elevate the level of abstraction, and (6) focus continuously on quality. The UP has a long and proven history and has clearly evolved (Ambler, 2005).

Table 15

*History of Unified Process*

| Year | Event |
|---|---|
| 1988 | Objectory v1.0 is created by Jacobson's Objectory AB company. Rational Unified Process and Enterprise Unified Process came out from the Objectory process. |
| 1996 | Rational Objectory Process (ROP) 4.0 is created. Iterative concept is introduced. |
| 1998 | ROP is renamed into Rational Unified Process and RUP 5.0 is released. |
| 1999 | Rational Unified Process 5.5 is released with an enhancement of real-time and web-based development. |
| 2000 | Rational Unified Process 2000 is developed with the addition of business engineering techniques to the business modeling discipline and a more enhanced requirements approach. |
| 2003 | Rational Unified Process 2003 is released with an enhanced test discipline. |
| 2004 | Enterprise Unified Process is developed with the expansion of the enterprise management discipline. |
| 2005 | Agile Unified Process is developed |

Table 15 depicts how the UP has evolved through several variations. Among the many different versions, Rational Unified Process (RUP) 2003 was utilized in this study as a framework. As shown in Table 15, RUP came from the Objectory process v1.0 and evolved into RUP 2003 with various additions and enhancements. More recently, a lighter UP called the Agile Unified Process (AUP) was developed for an agile software

development. The AUP utilizes a streamlined approach, which has fewer activities and deliverables with a simplified method.

The structure of RUP is two dimensional: phases and disciplines. The phases represent the four major stages that a project goes through over time. The major stages include inception, elaboration, construction, and transition. The disciplines represent the logical activities that take place throughout the project. The disciplines are divided into main disciplines and support disciplines. The main disciplines include business modeling, requirements, analysis and design, implement, testing, and deployment. The support disciplines include configuration and change management, project management, and environment. Table 16 shows the two dimensions of RUP.

Table 16

*Two Dimensions of RUP*

| Dimensions | | RUP |
|---|---|---|
| Phases | | • Inception<br>• Elaboration<br>• Construction<br>• Transition |
| Disciplines | Main Disciplines | • Business Modeling<br>• Requirements<br>• Analysis & Design<br>• Implement<br>• Testing<br>• Deployment |
| | Support Disciplines | • Configuration & Change Management<br>• Project Management<br>• Environment |

The nine disciplines can be employed across two or more phases during the RUP development life cycle. For example, the business modeling discipline can be utilized in

both the inception and elaboration phases to understand the business environment. The requirements, the design, the implementation, and the testing disciplines can be employed across all four phases to classify requirements that the system must implement, to design a solution for the system that satisfies the requirements, to write code that makes the system actually work, and to conduct unit and integrated system testing. The deployment discipline can occur during the elaboration, construction, and transition phases to place a portion of the system, or the full system, into operation for users. The support disciplines can also occur across all four phases for planning and controlling the project. Table 17 shows where the nine disciplines are used within the four phases, and also where disciplines are mostly utilized. The next section describes the main objectives and activities of each RUP phase, based on the explanations of Ambler (2005) and Satzinger et al. (2005).

Table 17

*Utilization of Disciplines in RUP phases*

| RUP Disciplines | Phases |
|---|---|
| Business Modeling | Inception*, Elaboration |
| Requirements | Inception, Elaboration*, Construction, Transition |
| Analysis & Design | Inception, Elaboration*, Construction, Transition |
| Implement | Inception, Elaboration, Construction*, Transition |
| Testing | Elaboration, Construction*, Transition |
| Deployment | Elaboration, Construction, Transition* |
| Configuration & Change Management | Inception, Elaboration, Construction*, Transition |

| RUP Disciplines | Phases |
|---|---|
| Project Management | Inception, Elaboration*, Construction, Transition |
| Environment | Inception, Elaboration*, Construction, Transition |

(* represents the most utilized phase for a certain discipline)

*The Inception Phase*

The primary objectives of the inception phase are to (1) identify the business scope of the new system and the project, (2) develop preliminary cost and schedule estimates based on the stakeholder concurrence, (3) identify the business need for the project, (4) understand the requirements according to the business case for the project, and (5) establish a vision for the solution. As shown in Table 17, the business modeling discipline is highly utilized in the inception phase. The main activities of the business modeling discipline in this phase include: (1) create a list of business benefits, system objectives, and system capabilities, (2) describe the problem or need, (3) consider business process, workflow, and interfaces to other systems, and (4) analyze the various system stakeholders, existing system architecture, and system constraints.

*The Elaboration Phase*

During the elaboration phase, detailed information is gathered, hence the requirements discipline is mostly utilized in this phase. Based on the gathered information, functional and non-functional requirements are defined. The functional requirements are activities and processes that the news system should carry out. The non-functional requirements are characteristics of the new system other than the activities it must perform. Some of non-functional requirements can include technical requirements, performance requirements, usability requirements, reliability requirements, and security

requirements. All defined requirements are prioritized and evaluated with actual system users. A structured walkthrough with users is an important process to make sure the gathered and prioritized requirements are correct and appropriate. User interface dialogs can also be developed in this phase.

*The Construction Phase*

The main focus in this phase goes to coding and testing the software. All the system components and features, including user interfaces, business logics, data access functions, and help functions, are implemented according to the specifications designed in the previous phase. This phase should produce a releasable working system so that the system can be deployed during the next phase. This phase can include several iterations that continue the design and implementation of the system. In particular, for large projects, several construction iterations can be involved in an effort to break the project into small and manageable tasks.

*The Transition Phase*

During the transition phase, the system is delivered into production and becomes available to end users. One or more iterations in this phase should involve end-user training with a user's manual, beta testing to validate the system functions against end-users' expectation, and corresponding modification and fine tuning. If all the requirements are satisfied, the development cycle is closed.

## *Quantitative Versus Qualitative Research Methods*

Though various research methods exist, the most commonly used classification of research methods is between quantitative and qualitative research methods. Quantitative

methods start with an assumption that variables can be identified and relationships can be measured. Quantitative methods usually employ theories, hypotheses, and mathematic models to generalize research findings to other people and places. Data collected in quantitative methods are numeric indices that can be quantified for statistical analysis. Generalization, causal explanation, and prediction are the main purposes of a research conducted with quantitative methods. In contrast, qualitative methods seldom begin with hypotheses and theories, and assume that variables are complex, interwoven, and difficult to measure. Qualitative methods employ non-numeric data, such as field notes, interviews, conversation, participant observations, questionnaires, documents and texts, photographs, recordings, memos, and researcher's reflection and reaction (Denzin & Lincoln, 2000; Glesne, 2006; Myers, 2009). Qualitative researchers use multiple data collection methods, called "triangulation," to increase the trustworthiness of the data (Glesne; Myers). The use of multiple methods can also secure an in-depth understanding of the phenomenon in question (Denzin & Lincoln). Qualitative research can be found in many disciplines and fields using a variety of approaches, methods, and techniques. Qualitative research is an interdisciplinary, and sometimes counterdisciplinary, field (Nelson, Treichler, & Grossberg, 1992). A qualitative researcher may be seen as bricoleur, a jack-of-all-trades, a kind of professional do-it-yourselfer (Lévi-Strauss, 1966).

Qualitative research uses a small number of samples, rather than a large number of random samples. The data is then categorized into patterns as the primary basis for organizing and reporting results. Quantitative data can be measured, while qualitative

data normally cannot be put directly into contexts that can be graphed or displayed by mathematical terms.

Table 18

*Differences between Quantitative and Qualitative Method (Note. From Becoming Qualitative Researchers, by Glesne, C, 2006, Boston: Pearson. Copyright 2006 by Pearson Education Inc. Adapted with permission)*

| | Quantitative Methods | Qualitative Methods |
|---|---|---|
| Assumptions | • Social facts have an objective reality<br>• Variables can be indentified and relationships measured | • Reality is socially constructed<br>• Variables are complex, interwoven, and difficult to measure |
| Research Purposes | • Generalizability<br>• Causal explanations<br>• Prediction | • Contextualization<br>• Understanding<br>• Interpretation |
| Research Approach | • Begins with hypothesis and theory<br>• Uses formal instruments<br>• Experimental<br>• Deductive<br>• Component analysis<br>• Seeks the norm<br>• Reduces data to numerical indices<br>• Uses abstract language in write-up | • May result in hypothesis and theory<br>• Researcher as instrument<br>• Naturalistic<br>• Inductive<br>• Searches for patterns<br>• Seeks pluralism, complexity<br>• Makes minor use of numerical indices<br>• Descriptive write-up |
| Research Role | • Detachment<br>• Objective portrayal | • Personal involvement<br>• Empathic understanding |

Converting contextual data to quantifiable numeric data results in the loss of a large portion of the understanding of the phenomenon from the point of view of the participant and its particular social and institutional context (Kaplan & Maxwell, 1994). Quantitative research focuses mainly on numbers, whereas qualitative research focuses

on context. Myers (2009) argued that losing many of the social and cultural aspects of organizations is a major disadvantage of quantitative research. He also mentioned that "the quantitative researcher trades context for the ability to generalize across a population" (2009, p. 9).

Table 19

*Strengths of Qualitative Methods*

| # | Merits |
|---|---|
| 1. | Qualitative research is useful for describing the complex phenomenon |
| 2. | Qualitative research can conduct cross-case comparisons and analysis |
| 3. | Qualitative research describes in rich detail phenomenon as they are situated and embedded in local context |
| 4. | Qualitative research is useful for describing the complex phenomenon |
| 5. | Qualitative research is useful for studying a number of limited cases in depth |
| 6. | Qualitative research provides individual case information |
| 7. | Qualitative research is useful for describing the complex phenomenon |
| 8. | Qualitative research can conduct cross-case comparisons and analysis |
| 9. | Qualitative research describes in rich detail phenomenon as they are situated and embedded in local context |
| 10. | Qualitative research provides description and understanding of people's personal experience of phenomenon (i.e., emic or insiders' viewpoint) |
| 11. | Qualitative research determines how participants interpret the constructs |
| 12. | Data are usually collected in naturalistic settings |
| 13. | Qualitative research is responsive for local situation and conditions |
| 14. | Qualitative researcher is responsive for the change that might occur during the study |
| 15. | Qualitative uses an important case to vividly demonstrate the phenomenon to readers of a report |
| 16. | Qualitative research determines idiographic causation (i.e., determination of cause of particular events) |

Table 18 displays differences between quantitative methods and qualitative methods. There are many different qualitative research approaches including action research, case study research, anthropology, ethnography, grounded theory, hermeneutics, semiotics, and phenomenology. All these qualitative research methods

have a number of strengths, as shown in Table 19. Qualitative research has also several weaknesses as shown in Table 20.

Table 20

*Weakness of Qualitative Methods*

| # | Weaknesses |
|---|---|
| 1. | Knowledge produced might not be generalized to other people in other places, and at other times (i.e., the research results are applied to only particular people, in particular places, and at particular time) |
| 2. | It is difficult to make quantitative prediction |
| 3. | It is more difficult to test hypothesis and theory with a large participant pool |
| 4. | It takes more time to collect data compare to quantitative methods |
| 5. | Data analysis is often time-consuming |
| 6. | Research results can be easily influenced by researcher's personal biases and idiosyncrasies |

## *Qualitative Research Methods in Information Systems*

Quantitative research methods have been dominant research methods in Management Information systems (MIS). However, some prominent Information Systems (IS) researchers recently recognized the strengths of qualitative research methods and started to apply them to their research efforts. The interest on the qualitative methods has increased among many research communities due to the dissatisfaction with the type of research information produced by quantitative methods (Van Maanen, 1982). As identified by Benbasat, Goldstein, and Mead (1987), the roots of the dissatisfaction in quantitative methods include (1) complexity of multivariate methods, (2) the distribution restrictions inherent in the use of quantitative methods, (3) the large number of samples required, and (4) difficulty in understanding and interpreting the research outcomes generated through complex quantitative methods. Another reason why IS researchers

turned their eyes to qualitative methods can be found in the research results published by Baroudi and Orlikowski (1989). They conducted extensive research to discover the statistical power of papers published in leading journals in MIS. The statistical power is an important measure to any research using statistical inference testing. Their research survey reveals that the statistical power of those papers is, on average, substantially below accepted norms. Surprisingly, Baroudi and Orlikowski claimed that researchers in MIS typically have a 40% chance of not recognizing the observable fact under study, even though it, in fact, may exist.

There is much evidence that qualitative methods are considered as an appropriate method in MIS research. In 1999, MIS Quarterly, one of leading MIS journals, published a special issue on intensive research in information systems using qualitative, interpretive, and case methods to study information technology. Markus and Lee (1999) employed the term “intensive research” originally suggested by Weick (1984) to signal the variety of methods that are commonly called qualitative research. As a result of this special issue, nine articles related to qualitative methods were published. More recently, MIS Quarterly published another special issue on action research in information systems in September, 2004. In the forward of that special issue, Baskerville and Myers (2004) believed that action research methods could improve practical relevance in IS research. The lack of relevancy to practice found in many papers published in leading MIS journals has been a big issue. Benbasat and Zumud (1999) addressed this issue and provided many good suggestions to overcoming the lack of relevance to practice. It is noticeable that qualitative methods are considered by prominent MIS researchers to be valid methods to provide practical knowledge.

In 1997, Myers (1997) established a website to support novice and experienced qualitative researchers in information systems. The qualitative research at that time was still new within the information system field, so Myers thought creating a living scholarly resource would provide a focal point for the emerging qualitative research community. His website is filled with abundant information on how to conduct, evaluate, and publish qualitative research. In 2000, the International Conference on Information System (ICIS) held a panel discussion to assess the merits of Markus' article (1983), which was one of the most cited empirical examples of qualitative research in information systems. More recently, the ICIS held in Milwaukee in December 2006 also established one separate section just for qualitative research methods and provided panel discussions on qualitative methods.

As mentioned earlier, there are various qualitative methods. Among them, four qualitative research methods will be discussed here in terms of how they are utilized in the MIS field.

*Case Study Research*

According to the study of Orlikowski and Baroudi (1991), case studies are the most common qualitative method. Their study reveals that over 90% of 155 information systems research articles published between 1983 and 1988 in leading MIS journals (Communications of the ACM, MIS Quarterly, Proceedings of International Conference on Information Systems, and Management Science) employed case studies, laboratory experiments, or surveys. Among the three categories, the case studies account for 13% of the whole papers. Alavi and Carson (1992) also examined and analyzed 908 MIS articles published from 1968 to 1988 in eight core journals (Communications of the ACM, Data

Base, Decision Science, Harvard Business Review, Journal of Management Information Systems, MIS Quarterly, Management Science, and Sloan Management Review). Their research results showed that 146 articles or 33.4% of the sample were field studies and among them, case studies account for a total of 40 articles. Some of MIS papers that discuss or employ case studies are listed in Appendix H.

*Action Research*

Baskerville and Wood-Harper (1996) explain how IS researchers might employ action research in MIS. Avison, Lau, Myers, and Nielsen (1999) argue that researchers should try out their theories with practitioners to make academic research more relevant. A recent paper written by Fruhling and De Vreede (2006) demonstrates how action research can be utilized in a real situation and in a real organization. A paper published by Ytterstad, Akselsen, Svendsen, and Watson (1996) also shows a good empirical example of action research. Appendix H lists some of papers related to action research.

*Ethnography*

Ethnography has become more broadly employed in the organization study in information systems since early pioneering work by Wynn (1979), Suchman (1987), and Zuboff (1988). Orlikowski (1991) applied the ethnographic method to study how information technology deployed in work processes facilitates changes in forms of control and forms of organizing. Preston (1991) also examined the problems in and of management information systems using the ethnographic method. Davies and Nielsen (1992) discussed the ethnographic study of configuration management and documentation practices in an information technology center. Myers (1999, p.1) has argued that “ethnographic research is well suited to providing information systems

researchers with rich insight into the human, social, and organizational aspects of information systems." Appendix H shows more published papers in the area of ethnographic study.

*Grounded Theory*

Orlikowski's (1993) paper is considered one of the best examples of grounded theory in information systems. This paper was recognized as the best MIS Quarterly paper of the year for 1993. In this paper, Orlikowski used a grounded theory research approach to develop theoretical framework for conceptualizing the organizational issues around the adoption and use of CASE (Computer Aided Software Engineering) tools. Due to the usefulness in developing context-based, process oriented descriptions and explanations of the phenomenon, grounded theory approaches are becoming common in information systems research. Some of papers related to grounded theory are listed in Appendix H.

# CHAPTER III

# RESEARCH METHODS AND PROCEDURES

## *Introduction*

In this study, data was collected and analyzed using a qualitative method. In particular, case study research was utilized as the research method and grounded theory as the mode for analysis. After the rationale for selecting the case study research is described, the type of case study and the unit of analysis are explained, followed by a description of the site selection process, and an explanation of the data sources. Finally, the methods and procedures for the case study research and grounded theory are presented.

## *Rationale for Selecting Case Study Research*

As mentioned in the introduction section, two in-depth case studies were employed as a research method for two primary reasons. First, a case study has the capability of scrutinizing a phenomenon in its natural settings and utilizing multiple data collection methods to gather information from one or more people, groups, or organizations (Benbasat & McFarlan, 1984; Bonoma, 1985; Kaplan, 1985; Stone, 1978; Yin, 1989a). Second, a case study is known to be a well-suited method for capturing the knowledge of practitioners and developing theories from it (Benbasat et al., 1987).

There are three well-known reasons why case study research can be considered as a viable information systems research method (Benbasat et al., 1987). First, the research is carried out in a natural setting, the current state of the art can be learned, and theories can be generated from practice. Second, “how” and “why” questions can be answered;

that is, the nature and complexity of the process taking place can be understood. Third, a case study is an appropriate way to research an area in which few previous studies have been done. Based on the literature review, employing a case study is very appropriate to explore the issues and challenges of the Scrum software development method, where research and theories are at the early formative stages.

### *Type of Case Study Research*

There are at least six main different types of case studies (Yin, 1993). The case study research can be conducted with a single case or multiple cases. A single case study concentrates on a single organization or situation, whereas a multiple case study includes two or more instances within the same study. Further, a case study can be categorized as an exploratory, explanatory, or descriptive study regardless of single or multiple cases. An exploratory case study is typically employed to define the questions and hypotheses for a subsequent study or to determine the feasibility of the research procedures. An explanatory case study is used to identify cause-effect relationships. A descriptive case study illustrates a case study may fall into one of at least six (3x2) basic types of case studies as shown in Table 21. In this study, multiple-cases (two organizations) with an exploratory and a descriptive point of view were selected.

Table 21

*Six Different Types of Case Studies*

| Type | Single-Case | Multiple-Case |
|---|---|---|
| Exploratory | Exploratory with Single-Case | Exploratory with Multiple-Case |
| Explanatory | Explanatory with Single-Case | Explanatory with Multiple-Case |
| Descriptive | Descriptive with Single-Case | Descriptive with Multiple-Case |

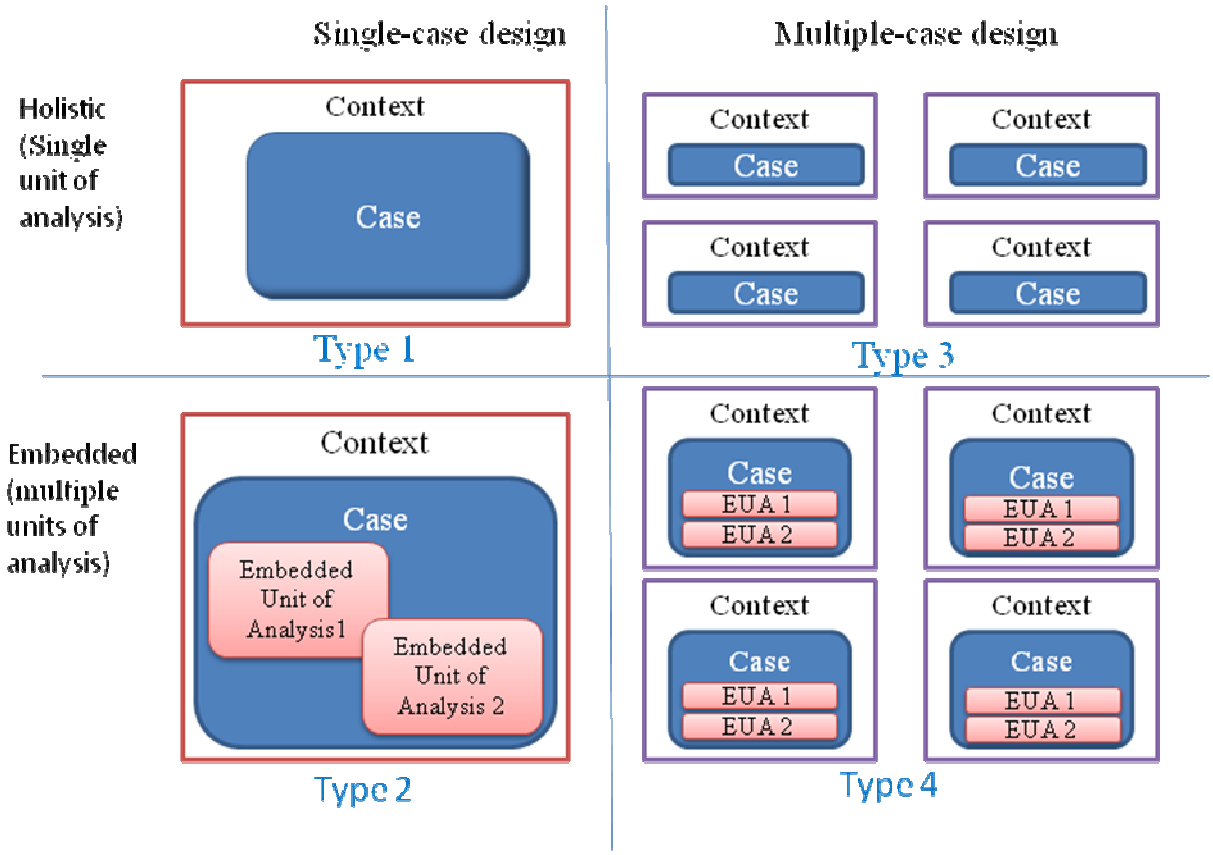



*Figure 7*. Basic types of design for case studies.
(Source: Yin, 203, p. 40)

*Unit of Analysis in Case Study*

Defining a unit of analysis within a particular case is critical both in the design and analysis phase of the case study (Yin, 2003). In the design phase, a well-defined unit of analysis can give researchers boundaries for the study which lead to an appropriate literature review and adequate data collection. In the analysis phase, research findings can be generalized to specific theoretic propositions about the defined unit of analysis. In addition, the unit of analysis becomes the main analytic level for the case being studied. Yin has suggested four different types of designs for case studies (see Figure 7). Both single-case and multiple-case study can have a unitary unit or multiple units of analysis.

Thus, the four types of suggested case studies are single-case holistic designs (Type 1), single-case embedded designs (Type 2), multiple-case holistic designs (Type 3), and multiple-case embedded designs (Type 4). Yin also mentions that there are five rationales for launching single-case designs including (1) the case represents the critical case in testing a well-formulated theory, (2) the case represents an extreme case or a unique case, (3) the case is representative or typical case, (4) the case is revelatory case, i.e., few researchers had previously investigated the case, and (5) the case is the longitudinal case (studying the same single case at two or more different points in time).

In this study, the Type 3 design (multiple-case with a single unit) was used. Thus the main unit of analysis was the entire software development process of Scrum (see Figure 8). This main unit will include all of the organizational and technical activities taking place over time.

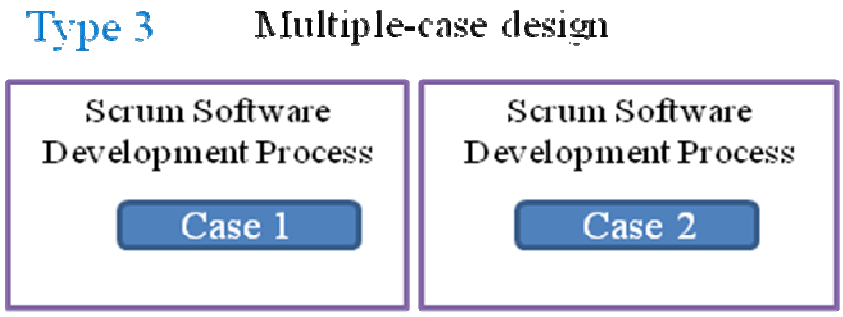



*Figure 8*. Multi-case design used for the research.

*Site Selection*

Two research sites were selected for their similarities as well as their differences, based on the technique of theoretical sampling (Glaser & Strauss, 1967). Both

organizations chosen for this study have used Scrum for the past few years in their systems development processes. The Scrum method in both organizations was integrated in every aspect of their software development processes, including planning, analysis, design, coding, and testing. While one organization has been providing large-scale and mission-critical applications, the other organization has been providing small- and medium-scale applications. The two organizations also differ on size, industry, and location. These differences between the organizations provide useful contrasts to be made during data analysis. One organization produces large-scale and mission-critical applications will be called the ABC firm, while the other organization will be called the XYZ firm. The next section briefly describes the background of these two firms.

*ABC Firm*

The ABC firm has been providing mission-critical public safety software to police departments, fire departments, 911 dispatch centers, sheriff's offices, and airport authorities since 1978. Due to the nature of software related to the public safety, a small glitch in the company's software can cause severe disasters or casualties. The ABC firm has strived to meet the highest standard of robustness and reliability because of their great impacts on the public safety. As of this paper, nearly 600 agencies and over 30,000 public safety officials in the United States use the company's software, which includes a records management system, a computer-aided dispatch system, a fire/emergency medical service management system, and a jail management system.

The jail management system was recently re-designed and developed using the Scrum method, with five to eight software engineers working for more than two years. The jail management system was mostly developed using the C sharp (C#) computer

programming language on the Microsoft .Net platform for front-end graphical user interfaces and C UNIX programming language for the management of a UNIX-side database system. The jail system consists of more than a million lines of code. The company has a total of 30 software developers, product managers, and Quality Assurance (QA) personnel in the software development division and has been using Scrum since April, 2005.

About four months after the company decided to use the Scrum method, the main part of the company moved to a different city which is about 100 miles away from the original site. The company kept the two work places for about a year, with the software development team divided between the two locations.

*XYZ Firm*

The XYZ firm has been providing internet-based database applications primarily to clients in the government sector for over fifteen years, with products in vital statistics/records, environmental water quality and human services. Some current clients include the Environmental Protection Agency, the Federal Aviation Administration (FAA), the States of Arizona and Montana, and the District of Columbia. The firm has been using the Scrum software development method and has successfully completed several projects, including a vital statistics, electronic birth and death registries system, a Safety Programs Airmen Notification System (SPANS), and an emergency medical system.

SPANS, which was built using Microsoft's .NET framework, was re-designed and developed to assist the Federal Aviation Administration Safety Team (FAAST) in meeting their goals of providing airmen with safety program information and notification

in the timeliest manner possible, while providing an easy way for airmen to make suggestions to the safety program for needed information. SPANS also allows the FAA to keep in touch with airmen and alert them to safety seminars, events, and important changes that affect them, both regionally and nationally.

*Comparison of Two Firms*

As discussed previously, ABC and XYZ operate in different markets. The duration of a project at ABC firm is longer than one at XYZ, with the average duration of projects at ABC being 1-2 year(s) and at XYZ 3-6 months. ABC utilizes various computer programming languages, such as Java, C, C++, C#, and Perl, on both UNIX and Windows operating systems. XYZ mainly uses Java and HTML-based web programming languages on the Windows platform. Table 22 summarizes various differences between the firms.

## *Data Sources*

In an attempt to gather data, three types of data were collected from both firms to triangulate findings and enhance trustworthiness (Gall, Gall, & Borg, 2003; Glesne, 2006). First, observations of software development process was conducted through on-site visits and field notes were taken during the observation to make the strange familiar (Erickson, 1973). Second, an email survey was developed and conducted among software developers, QA personnel, and managers. The survey instrument was mainly used to refine interview questions. Finally, a formal face-to-face interview was conducted with executive officers, project managers, lead software engineers, and developers. All of the formal interviews were audio-taped, transcribed, and later coded for analysis. This

triangulation in the process of data collection provides more useful information and different perspectives on the issues, allows for cross-checking, and yields stronger substantiation of constructs (Eisenhardt, 1989; Glaser & Strauss, 1967; Pettigrew, 1990).

Table 22

*Differences Between ABC and XYZ Firm*

| **Categories** | **ABC Firm** | **XYZ Firm** |
|---|---|---|
| Main Applications | • Jail Management System,<br>• Computer Aided Dispatch System<br>• Fire/Emergency Medical Service System<br>• Records Management System | • Vital Statistics/Record System<br>• Environmental Water Service System<br>• Human Service System |
| Size of Projects | Large-Scale | Small/Medium-Scale |
| Mission-critical Projects | All applications are mission-critical | Some of applications are mission-critical |
| Average Duration of Projects | 1-2 year(s) | 3-6 months |
| Computer Languages | Java, C, C++, C#, Perl | Java, HTML |
| Development Platform | Unix and Windows System | Windows System |

Data collection focused on issues and challenges of Scrum. It sought information on: empirical process control (visibility, inspection, and adaptation) in Scrum; the roles of the product owner, team members, and Scrum master; the daily Scrum meeting, the daily Scrum of Scrums meeting, the Sprint review, and the Sprint planning meeting; the product backlog, the Sprint backlog, and burndown charts; the flow of Scrum process; user involvement; training; documentation; communication; individual and team

experience with Scrum; applicability of Scrum; bug tracking system; project estimation and planning; and the working environment.

The first field study was conducted within the ABC firm. During the field study, which lasted for about 8 months, daily systems development activities under the Scrum process, such as the daily Scrum meeting, the Scrum of Scrums meeting, the Sprint planning meeting, and the Sprint review meeting, were observed. In addition to the observation, an email survey was conducted among 15 people including developers, Quality Assurance (QA) personnel, and project managers. The following open questions were used as survey questions:

1. What have worked and have not worked for you on a project since you began using Scrum?
2. What are the most unique and interesting aspects of Scrum?
3. What have you learned from Scrum?
4. What would you do next time with Scrum?
5. What advices do you have for others involved in Scrum?
6. Do you have any comments or opinions on any of the following Scrum process?
    a. Daily Scrum meeting
    b. Daily Scrum of Scrums
    c. Sprint planning meeting
    d. Sprint review meeting
    e. Product backlog
    f. Sprint backlog
    g. Scrum master

Some of selected project team members, including developers, lead engineers, project managers, and executive officers were interviewed both informally and formally. Each informal interview took an average of a half an hour and summary notes were taken. The formal interviews took an average of one hour and were audio-taped and transcribed later. Table 23 shows the number of people, type, and amount of interviews conducted at the ABC firm.

Table 23

*Type and Number of Interviews Conducted at ABC Firm*

| Position (number of people) | Informal Interview | Formal Interview | Total |
|---|---|---|---|
| VP in R & D Department (1) | 2 | 1 | 3 |
| Project Manager (2) | 5 | 2 | 7 |
| Lead Engineers (2) | 4 | 2 | 6 |
| Developers (7) | 14 | 5 | 19 |
| **Total (12)** | **25** | **10** | **35** |

The second field study was conducted within the XYZ firm. The field study, which started with an informal interview with an executive officer, took about 4 months. The Scrum processes in 5 Scrum teams were observed and an email survey was conducted among developers, QA personnel, and project managers. The same survey questions used at ABC were utilized. Informal and formal interviews were conducted among 10 people including developers, project managers, a director of operations, and a senior vice president of operations. Table 24 shows number of people, type, and amount of interviews conducted at XYZ.

At each site, the observation was conducted in both observe-only mode and in a mode where a participant observer was allowed to talk. Various levels of individuals, such as developers, lead engineers, project managers, and executive officers, were selected to provide data from multiple levels and perspectives. This was done to answer Leonard-Barton (1990, p. 249), “In order to understand all the interacting factors, it is necessary that the research methodology slice vertically through organization, obtaining data from multiple levels and perspectives.”

Table 24

*Type and Number of Interviews Conducted at XYZ Firm*

| Position (number of people) | Informal Interview | Formal Interview | Total |
|---|---|---|---|
| Sr. VP in Operations (1) | 2 | 1 | 3 |
| Director of operations (1) | 2 | 1 | 3 |
| Project Manager (2) | 3 | 2 | 5 |
| Developers (6) | 6 | 6 | 10 |
| **Total (10)** | **11** | **10** | **21** |

While the primary unit of analysis was the Scrum software development process, the collection of inter-related data at other levels of analysis (Pettigrew, 1990; Yin, 1989a, 1989b) was considered. All the formal interview questions were semi-structured, and the complete list of interview questions was listed in Appendix E.

### *Grounded Theory*

In the process of data analysis, grounded theory (Glaser & Strauss, 1967; Martin & Turner, 1986; Turner, 1983) was employed with an aim of generating descriptive and explanatory theory associated with Scrum software development process. This approach has been effectively utilized in organizational research (Anacon, 1990; Elsbach & Sutton, 1992; Isabella, 1990; Kahn, 1990; Pettigrew, 1990; Sutton, 1987). Grounded theory is "an approach to theory development that involves deriving constructs and laws directly from the immediate data that the researcher has collected rather than drawing on an existing theory" (Gall et al., 2003, p. 626). Sometimes, it would be better for researchers to collect data and analyze them without conducting a review of the literature beforehand

(Glaser, 1978). The main purpose of grounded theory, which was initially developed by Straus and Corbin (1988), is to “demonstrate relations between conceptual categories and to specify the conditions under which theoretical relationships emerge, change, or are maintained” (Charmaz, 2002, p. 675).

# CHAPTER IV

# DATA ANALYSIS AND RESEARCH RESULTS

This chapter describes the process of data analysis and empirical findings through grounded theory study of two firms that implemented Scrum in their software development processes. Concepts suggested by the open coding of raw data and recurring themes identified from axial coding are also explained.

## *The Process of Data Analysis*

In the first stage of data analysis, all the data produced by observations, email surveys, documentations and interviews were examined and coded by focusing on the issues and challenges of Scrum identified using the empirical processes controls and the Scrum framework (roles, ceremonies, and artifact). In the first round of data analysis, an open coding technique (Straus & Corbin, 1990) was used to identify possible concepts, along with their properties and dimensions. The open coding technique involves a form of content analysis where the data is read and categorized into concepts that are suggested by the data rather than imposed from the outside (Agar, 1980).

In the second round, the codes were reviewed, and the concepts were organized into recurring themes. These themes were used later as a basis for creating a set of stable and common categories. During the second round of data analysis, the documents were re-read, and analytic and self-reflective memos were created. The final stage of data analysis was completed through an axial coding (Straus & Corbin, 1990) which depends on a synthetic technique of making connections between categories and subcategories to build a more comprehensive scheme. During this stage, all the codes were searched,

sorted, grouped, and compared to the original data. This process continued until a final series of categories were identified, each having a high frequency of occurrence in the data. Figure 9 illustrates the process of the data analysis. In the following section, each of the concepts and categories found in the ABC and XYZ firm is presented and discussed.

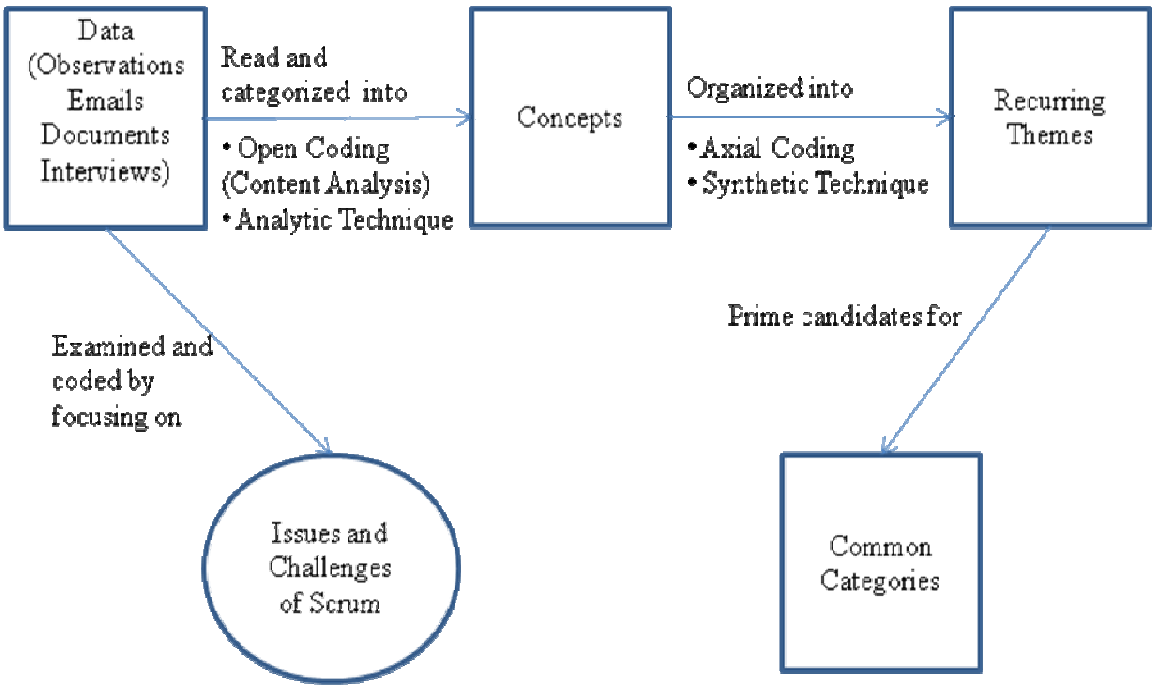



*Figure 9*. Data analysis process.

*ABC Firm*

ABC has about 30 software engineers in five development teams in the software division. ABC used a waterfall style traditional software development method for more than 20 years before they changed to the Scrum method. To test the success of Scrum method, all software engineers in the firm were invited to 2 days-long, 8 hours intensive Scrum training sessions, which were conducted by one of leading training experts in this

field (AgileLogic, 2006). The firm then reorganized their development teams following the Scrum model. Each team consists of four to seven software engineers, a Quality Assurance (QA) person, a Scrum master, and a Product Line Manager (PLM). These teams are specialized in the area of interface, jail, record, mobile, and architecture.

When Scrum was first introduced to the firm, most developers were reluctant to adopt a new method due to their previous experience with traditional development methods. However, they began to enjoy several unique features of Scrum as they exercised its' various elements. In general, most elements of Scrum were successfully adopted and implemented with minor modifications dealing with meeting schedules and the open working environment. The following sections explain sixteen concepts that appeared in the data analysis process.

Table 25 lists the categories, concepts, and data related to ABC. As shown in the table, the factors of human resource management, structured development process, environmental, and information systems and technology were constructed. The concepts comprising these categories are discussed in the following section.

*Human Resource Management Factor*

The concepts of team management, collaboration, training, lack of accountability, and trust and confidence constitute the human resource management factor. Each concept is discussed in turn.

Table 25

*Categories, Concepts, and Data Related to ABC*

| No. | Common Categories | Concepts | Data |
|---|---|---|---|
| 1 | Human Resource Management Factor | Team Management | • Teams were reorganized without considering developers' knowledge and skills<br>• Self-managing team does not work well<br>• Team needs a supervisor who can see a big picture<br>• Small team size is more flexible and adaptable in defining and applying a variant of Scrum |
| | | Collaboration | • Developers and QA does not collaborate<br>• Developers do not try to get to know new developers in remote site<br>• Several teams simultaneously work on "look and feel" design that should be implemented by a designated team<br>• Tools are used to reduce collaboration problems |
| | | Training | • New employee training is one of the biggest problems<br>• Because of the complex nature of the system, new employees need to spend a lot of time to be trained<br>• Employee training problem gets worse when two development sites are involved |
| | | Lack of Accountability | • Tasks in a Sprint backlog do not get completed<br>• Nobody takes responsibility on delayed tasks<br>• Self-managing Scrum team provokes the lack of supervision<br>• Project managers do not have any authority to control developers |
| | | Trust and Confidence | • Scrum master does not remove developer's impediments<br>• Trust and confidence are lost when people do not complete their work<br>• It is important to know who you are talking to and what level of information they have |

(table continues)

| | | | |
|---|---|---|---|
| 2 | Structured Development Process Factor | Scrum Framework | • The Scrum model defines a product backlog, helps prioritize tasks, keep track of task assignments, and helps monitor task progress<br>• Scrum increases our communication in the team. There seems to be better team work using the SCRUM model.<br>• Daily Scrum meeting is utilized by team members to understand what other members are working on and identify obstacles to overcome<br>• Scrum master is selected from non-technical person<br>• A group of Scrum masters have a daily Scrum of Scrums meeting<br>• Daily Scrum meeting is held too often<br>• Sprint planning meeting occupies too much time |
| | | Unit and Integration Testing | • A QA person needs to wait until developers finish their coding<br>• Developers do not want to review the code that they finished during the previous Sprint<br>• QA people want developers to set aside some time for bugs<br>• QA people may not know what areas could be affected by the changes made by developers |
| | | Coding Standard | • Developers can understand other developer's code better through coding standards<br>• Too much coding standards may hamper developers performance |
| | | Documentation | • Detailed design documents were reduced significantly<br>• (Example of documents: use cases, class diagram, sequencing diagram, activity diagram, communication diagram)<br>• Quick implementation without creating a document can cause ripple effects that damage other parts of the project<br>• No one takes time to think about inter-dependency<br>• Bug rate is increased because of reduced documents which lead to the lack of standardization of features, field names, and error messages |

(table continues)

| | | | |
|---|---|---|---|
| | | Formal Code Review | • A web-based code review tool was created by a project manager<br>• Reviewers checked if there were any side and ripple effects to other code<br>Formal code review was a vital and critical process for high-quality applications |
| 3 | Environmental Factor | Customer Involvement | • Customers cannot participate in Scrum meetings due to the large number of customers scattered in the U.S.<br>• Project managers spend lots of hours to visit customers<br>• The firm hosts a users' conference where customers can vote for or against a policy and a direction of new application development<br>• QA personnel cannot provide customers with quick turn around on bug fixes |
| | | Working Environment | • Open working environment promotes communications, facilitates self-organization, and makes developers to get together easily<br>• We are constantly distracted by the person we work with in a cubicle setting<br>• Putting every team member in one cubicle area increases collaboration and teamwork |
| | | Interdependency among Modules | • QA department has found twice as many bugs since the firm switched to Scrum<br>• As the size of application grows, the dependencies and interconnections among tasks increase<br>• Developers have a tendency to complete a task in a quick and dirty way<br>• Developers do not think of how code will be maintained and the code will be flexible enough for future needs |

(table continues)

| | | | |
|---|---|---|---|
| | | Social Facilitation, Social Loafing, Group Motivation, and Evaluation Apprehension | • Social facilitation effect and group motivational gains were observed as factors that reduced development times and lowered bug rates<br>• Evaluation apprehension was observed when the company invited all of developers, QA personnel, and people in other to the Sprint review meeting |
| 4 | Information Systems and Technology Factor | Communication System | • The firm put a lot of efforts to establish a good communication channel<br>• Video conference equipment was used only for team meetings<br>• Phone conferencing did not provide facial expression, gesture, and body language<br>• A web demo tool together with a phone system worked very well<br>• Some of tools including instant messaging, email, virtual private networking, and remote desktop were also used<br>• Multimedia systems did not work as optimally as face-to-face conversation<br>• If data line is down employees cannot do anything |
| | | Information and Knowledge Sharing System | • Well-structured information and knowledge sharing systems are needed between experienced developers and brand new developers<br>• A web-based Wiki program is used to facilitate the information and knowledge sharing<br>• The Wiki program helps to mitigate the problem between two sites |
| | | Bug Tracking System and Management Tool | • A Unix-based bug tracking system called MOM is utilized by different departments<br>• The level of bug severity is assigned to each bug<br>• A commercial tool called JIRA is adopted<br>• JIRA is used to record defects found in alpha and beta testing and in standard code<br>• JIRA provides filter functions and severity priority codes that rank the level of severity of bugs<br>• JIRA does not provide a decent search engine |

*Team Management.* In terms of development times and costs, Scrum was not very effective during the first two months after ABC switched to the Scrum method. They attributed this undesirable fact to the inappropriate composition of the development teams. The teams were reorganized without considering knowledge and skills of the developers. Therefore, some team members had to learn business logic (how the application works in a specific field) that they were not familiar with, and others needed to learn new development tools and programming languages. As a result, it took longer development times and more development costs. A project manager stated:

> Teams cannot be thrown together. The team needs to be built. Without the right mix of team members Scum development is much slower. You can still succeed with Scum development. But without all members working together, building the team, the interest, and the enjoyment, you are not going to have the success and the energy from the team that the Scum development can bring.

Some developers dislike a self-managing team though this is identified as one of the unique aspects of Scrum. They prefer to have a team lead who can keep things going in the right direction. They think the Scrum master might be a good candidate to do this job. One developer added:

> I'm not sure that I like having the group of developers run the team. I would still prefer having a team lead type of person who can work with the team to help focus and keep things going in the right direction. I think the Scrum master should be doing this but I think this person needs to be more technical. I don't think this person should take the team over however, I think they should just help focus the team on the right priorities.

Another developer also mentioned that, "Our teams aren't very good at self-managing, we're more used to someone being in charge." A QA person revealed another possible problem with teams, developers given too much power to runs the team. He vented that "our problem was when team members didn't interact well with other members of the

team." One developer expressed that a self-managing team was strange to him, and he wanted to have a supervisor who can see the big picture. He stated:

> When we are going to take vacation/sick leave, we report to our team members rather than a manager. I'm not sure if that's good or bad, but it seems strange to me. It sometimes feels like we don't have much supervision, which I personally like, but we seem to lack a person or group that is trying to see the big picture.

However, most of developers consider a self-managing and self-organizing team a great idea. One developer stated that, "The part of Scrum that has worked the best for me is the idea that the team decides how to do things based on the consensus of the team." Another developer mentioned, "There seems to be better team work using the Scrum model. I like the idea that the team has more control over how the development is done and completed." A project manager added, "Team work is very important! The most interesting and unique part of Scrum has been better team work." The firm tried to keep the size of Scrum team as small as possible, believing that teams can be more flexible and adaptable in defining and applying an appropriate variant of Scrum. A vice president of software R&D department stated:

> It seems the Scrum method provides better team work through the Scrum framework. We keep each team as small as possible as recommended by the Scrum method, and it appears that a small Scrum team is more efficient when implementing various components of Scrum.

*Collaboration.* When ABC first adopted the Scrum method, five Scrum teams were organized and each team had a Quality Assurance (QA) person who was designated to test the code created by his/her own team. Recently, the company pulled the QA person from each Scrum team and created a QA personnel-only Scrum team so that a QA person can test the code generated by any Scrum teams. Because the QA Scrum team is located in one place, it sometimes causes collaboration problems between remote site

developers and QA personnel. For example, when developers in a remote site pass the code over to QA personnel through the Concurrent Versions System (CVS), QA personnel may not know which part of code is affected by the changes that developers made. A lead engineer noted:

> The development was kind of passed over to QA and they took care of testing. But the problem with that is QA may not know what particular area could be affected. So, they cannot test the area which might be broken by the changes made by developers.

This problem might be solved if developers take on some parts of QA's testing or show the QA personnel the potential areas that might be affected by the modification done.

Another collaboration problem arose when new developers of the Scrum team were hired in one location. Developers in the other location did not try to get to know the new developers. One developer put it this way:

> When new members of the team were hired at the other site, it was easier for team members here not to really put any effort into getting to know that person and learning how best to work with them. This problem was especially bad when the new members had some personality characteristics that made them a little annoying or perhaps difficult to develop a desire to want to work with them.

This issue mainly resulted from the way the Scrum team is divided at two different sites. It would be better for the company to reorganize the Scrum team so that team members are located at the same place.

One notable observation occurred at a daily Scrum of Scrums meeting with the firm's vice president of software development. Duplicates works were found across Scrum teams due to a lack of collaboration among the different teams. It was also noted that ABC had a difficult time keeping the product output consistent across teams. For example, several teams may simultaneously work on the "look and feel" design that

should be implemented by a designated team (e.g., an architecture team). This implies that there should be a person (possibly a Scrum master in each team) who is responsible for checking the consistency of products across Scrum teams. It was noted that the inconsistency and duplicates across different teams could negatively affect the efficient product management.

Other collaboration problems arose when developers and support staff discuss how to divide and assign tasks between the two sites in the Sprint planning meeting and how to track bugs reported by QA personnel and customers. To reduce the collaboration problems in this area, the firm has been using a web-based commercial tool called VersionOne (http://versionone.com), which has provided an excellent project management mechanism for both sites. Through VersionOne, developers at both sites can actually see how each project is divided, what projects are going on, the status of each project, who is working on each project, and when those projects are expected to be completed.

Recently, the company picked another commercial tool called JIRA (http://www.atlassian.com/software/jira/) to track information mainly on what kinds of bugs exist, who is working on each bug, and the status of each bug. JIRA actually replaced an old tool called MOM, which was developed internally and used to create information for support problems. However, using JIRA has some downsides. A project manager aired this issue:

> But it (JIRA) also has drawbacks. We've now broken up for instances, the fixed description, the problem, various fields, and who's worked on it. But in JIRA, you can't search. If you are looking for some word in the problem somewhere, you have to search each and every individual field. So, if you think, for instance, some value is in there in fields other than the comments, summary, and description, you

> must put the value in each field and then search individually. And searching is not near as good as MOM had. For instance, you can only search on whole words. You have to have the beginning of word. You can't search for the value within a word.

Despite the downsides, JIRA has been providing developers with a useful bug tracking mechanism for both sites. JIRA also has a special code, called the severity or priority code, which enables support people to automate the choice on what bugs to work on next. VersionOne and JIRA play a vital role in the company towards reducing collaboration problems that can arise in tracking and managing projects and bugs.

*Training.* New employee training issues emerged when information and knowledge-sharing issues and communication issues were brought up. A couple of project managers mentioned that one of the biggest problems that the company has been facing is new employee training. Because of the complex nature of the software program that the firm has been creating, new employees need to be trained for an extended period of time. A project manager noted that:

> due to the complexity of our application, new software engineers need to spend a good chunk of time to understand various aspects of the system. The typical employee training program usually lasts six month.

Another project manager added:

> "Right now, each Scrum team is in charge of training of new software engineers. This involves a selection of a mentor who leads and guides the new employee. The mentor spends a lot of his/her time with the new employee to provide the necessary knowledge and skills for projects. I think this is a big waste of time because the mentor in each Scrum team teaches some common parts of the system that can be taught together with other employees who belong to other Scrum teams."

The project manager suggested, “It would be more time-saving if one mentor teaches all new employees together the common parts of the system, and then let each Scrum team teach its new employee team-specific parts.”

It seems the problem gets worse when an employee who has expertise in one field is at the remote site and the new employee needs to learn the new field from the remote employee. A developer declared, “We often times have a situation where a new employee needs to know some parts of the system and the person who knows about them is at different location.” Another developer contributed, “We have some difficulties when we are involved with other team members, part of the team is here and part of the team is at the remote site. In particular, when we have a new member here and he/she needs to talk to other members at a different site.”

To address this problem, the company used multi-media, such as phone/video conferencing or web demos, but the training through these information technology has limitations. A developer pointed out, “We use a phone or a video set whenever a new employee needs to converse with an employee in remote site but it’s not efficient.” Another developer explains it this way, “You have to have a face-to-face and one-on-one training in order to make it efficient. You can do the training through the phone or video conferencing, but it’s not efficient.”

*Lack of Accountability.* One of the interesting aspects of the Scrum method is the ability to see the visibility of the progress of each project in every thirty or fifteen days, depending on the duration of the Sprint. In the Sprint planning meeting, team members divide items in the product backlog into a set of small and manageable tasks, which will

be entered into Sprint backlog. Based on the estimated completion time for each task, the set of all tasks in Sprint backlog is determined in such a way that they can be completed within a month. However, the tasks in the Sprint backlog usually do not get completed as estimated, and tasks are consistently carried over to the next Sprint Meeting. Moreover, nobody takes responsibility for that. A project manager noted, “Some tasks just keep getting carried over to the next month. There does not seem to be any responsibility on the developer side to complete the task. If they do not, it is o.k. and they hope to do better the next month in estimating their project.”

Another unique aspect of the Scrum method is self-managed teams, where the team is managed by the team members without having too much supervision from outside of the team. As part of the self-managing team, team members in the daily Scrum meeting choose tasks that they will work on for the day and report what has been done since the last daily Scrum meeting. However, this setting seems to be not working properly due to the team member’s lack of accountability on the tasks they chose. This seems to be a source of delayed projects. A project manager noted, “In the daily Scrum, there seems to be some lack of ‘what did you do yesterday?’ accountability. And taking on specific tasks for that day to work on is usually too generic. I believe we have become too relaxed on what we accept and that encourages some projects to drag over time.”

Another issue is that project managers do not have any authority to urge developers to work faster or harder. The managers do not take any accountability on delayed tasks because they do not have any control over that. The project manager contributed, “When we meet with the management team at the end of the month, no one seems to take the responsibility for those items not completed. The project managers do

not have any control on getting people to work faster or harder, so it is a little frustrating."

*Trust and Confidence.* Trust and confidence is an issue noticed between developers and Scrum masters. A Scrum master is supposed to do administrative work, mainly by helping developers focus on their work, by providing what the developers need, or by removing developer's roadblocks. It seems that developers badly want to have a Scrum master who does the job. However, one Scrum master was unable to get developers items or information they need and was unable to remove developer's impediments. He also did not follow up with developers to explain why he was unable to help them. A developer mentioned, "We need somebody who is going to be able to get us what we need and somebody telling us that 'You're going to get your information back.' And we weren't getting any information back and things weren't followed up." As developers do not get what they need from the Scrum master, they quickly lose their trust and confidence in the Scrum master. Another developer stated that, "I can't trust people who are not doing the job that they are supposed to do or that they said they are going to do. I lose my confidence when things don't get done."

Trust and confidence as an issue was also noticed between developers when developers worked together on the same project, and they did not see any progress on a module that was assigned to a developer. A developer mentioned that, "I usually work on a certain task with other developers, and sometimes I lose my trust if I don't see any progress from a task assigned to other developers."

The same issue arose if developers were divided into two development sites. For example, a Scrum team member asks a team member at the other site to do a certain task, but no progress on the task is visible. It seems developers at one site had a hard time establishing a feeling of trust in Scrum masters or developers at the other site when they do not complete tasks, and the lack of trust led to a reduction in confidence. As a developer stated, “People in one site didn’t trust people in the other site if they were not able to get the job done. I think that was one of major problems we had. That really hurts our confidence.” Another developer talks about the level of confidence and wants to know the level of information that other developer has. He noted that, “I need to be able to have a confidence level. I think it’s really important to know who you are talking to and what level of information they have.” He also mentioned, “I work better with people who I know better so I understand what they are really saying.”

*Structured Development Process Factor*

As shown in Table 25, the structured development process factor consists of the Scrum framework, unit and integration testing, coding standard, documentation, and formal code review. Each concept is presented in the following section.

*Scrum Framework.* The majority of developers are in favor of a Scrum framework. One developer expressed that, “I like how the Scrum model defines a product backlog, helps us prioritize tasks, keep track of task assignments, and helps us monitor task progress.” Another developer stated that, “Scrum increases our communication in the team. There seems to be better team work using the Scrum model. Working in the Scrum team provides motivation, excitement, and interest.” A project manager also

mentioned that, “I think the Scrum model gives us all a goal to strive for each month. This is motivating and helps us stay on task.”

In terms of daily Scrum meetings, a couple of teams had a standup meeting and the rest of teams had a seated meeting. However, the principal of short daily Scrum meeting time was well observed (less than 15 minutes) by all teams. Several developers claimed, “it was not necessary to have Scrum meeting everyday when there were no specific agenda to discuss”, but the majority of developers and QA personnel enjoyed this daily-based meeting. The daily Scrum meeting also seems to be a good time for team members to understand what other members were working on and identify obstacles to overcome. For example, if a developer says, “I am stuck with a certain problem,” other developers would typically respond with “I can help with that. Let’s get together after the meeting” or “I don’t know the solution for that but I can look at your problem after the meeting.” One developer noted that, “I like the daily status updates, especially on high priority tasks.” Another developer stated, “Through the daily Scrum meeting, team members are able to refine the goal for each Sprint and improve the quality of products.”

The firm appointed a Scrum master for each of the five development teams. Because the company believed that the main role of the Scrum master is to provide the administrative services for his or her team members, the company appointed non-technical persons as Scum masters. The non-technical Scrum masters sometimes seemed to be a problem for developers. One developer stated that, “Not having the correct Scrum master was a problem. Team impediments were not taken care of and often needed to be repeated, slowing or even stopping progress.” Another developer specified, “Our Scrum masters don’t have enough technical backgrounds to understand the things that we’re

dealing with. They sometimes had a hard time removing our roadblocks because of their lack of understanding of our technical stuff."

The firm also held the Sprint review meeting, called a "products fair". The products fair was unique in the sense that developers in all teams, QA personnel, and people in other department (sales, marketing, customer support) were strongly encouraged to attend. The firm expected that, given an opportunity to show their accomplishments during the Sprint, developers would diligently work to make sure that they are currently on the pre-determined development schedule with appropriate levels of intermediate deliverables of the products.

The Sprint planning meeting seemed to provide smaller and manageable tasks to developers. Several developers stated that, "I like the Sprint planning meeting where you have to breakdown the projects into smaller tasks. This makes it easy to manage big projects." Also, the product backlog and burndown chart appeared to help developers to organize and prioritize the schedule, and track the progress during the Sprint. Several developers expressed similar opinions, like "I like having product backlog reviews to organize and prioritize the schedule each month. I also like a burndown chart and its ability to track our progress during the month."

Though most of people liked the daily Scrum meeting, some of developers expressed their worries. One developer stated:

> We meet each day as opposed to weekly or bi-weekly. It is good to have feedback, but sometimes it feels like too much time for not enough value. I don't think it is necessary for that much reporting and it takes too much time away from programming. I think we could report 2 or 3 times a week and still have the productivity level we now have in our team.

Another developer also mentioned:

> "While it is important to deliver quality products in a timely manner, it is also important to not spend too much time in meetings that do not offer a good return on their investment. Meetings are an investment for any company and keeping a good balance requires input from all parties and honest evaluation of time spent."

Several developers expressed their worries on the Sprint meeting. One of them thought the meeting occupies too much time. He mentioned that, "on the negative side, I think it's a lot more overhead to have the Sprint meetings, which take all day at the end of each month."

Some QA personnel also wanted to have a streamlined planning session. One QA person stated that, "We need to streamline the planning session as much as possible. Going to meetings repeatedly is not productive." Finally, one developer didn't like Scrum at all. He mentioned that, "Frankly, I don't like Scrum, I don't think I'd do it given the choice. I would prefer to simply adopt some of the Scrum methods into current processes."

*Unit and Integration Testing.* When regular teams were reorganized into Scrum teams, a QA person was assigned to each Scrum team. However, the QA person in each Scrum team needs to wait until developers in the team finish their coding during the Sprint, which lasts usually for 30 days. This process causes a problem because the QA person in each team is always behind the Sprint schedule. A QA manager mentioned:

> I find that it's really hard to test code if there's no code to test. So, QA is always behind the rest of the team. The programmers finish coding within the Sprint, but QA isn't finished within the Sprint. So, QA is added to the next Sprint.

Another problem with this setting is that when QA people work on the code that developers just finished in the previous Sprint, they often times need to talk with developers about the code and developers do not want to go back to the code that they already checked in. One QA person stated that, “It's like pulling teeth to try to convince the team that I'm going to find bugs with their code that they just threw together.” QA people want developers to spend some time to talk about bugs with them. Developers just want to keep writing code instead of revisiting the code created during the previous Sprint. Another QA person mentioned that, “Developers really should set aside some time to address the bugs in a timely manner so that I can continue with testing.”

To mitigate the problem, ABC pulled a QA person from each Scrum team and created a QA-only Scrum team. The QA-only Scrum team covers all code created by other Scrum teams. It seems this change has improved the efficiency in testing because QA people usually have some code to test, and they do not need to wait for developers in one Scrum team to finish their code. Associated with this setting, QA people were usually in charge of both unit and integration testing. The firm changed the testing process in such a way that developers took responsibility for unit testing, because QA people sometimes do not have the detail picture of the particular area where a developer worked. An executive officer offered:

> QA people are asking for developers to do some testing. In the past, the development was kind of passed over to QA and they took care of testing. The problem with that is a QA person may not know what particular areas could be affected by the changes made by developers.

One developer also mentioned that, “If you know what could be affected, you may focus your testing just on those areas. But sometimes, QA people do not realize that there is

another area also affected by some of the changes." Another developer shared his personal experience:

> One time I made some changes in UU command which updates a unit. It turned out that my change broke UC command which updates call and broke other commands. But QA people did not do any tests on those areas because they did not know they were related.

Due to the issues described above, ABC asked developers to do some portion of the testing that QA people usually cover, but this may not a good move economically for ABC because developers usually get paid more than QA people.

*Coding Standards.* ABC has utilized coding standards. They have very specific coding standards in many areas to facilitate easily maintainable and expandable code. ABC's document on C# coding standards describes comprehensive rules and conventions that developers should follow. Table 26 displays two coding standard examples associated with headers and naming conventions. Other coding standard examples are listed in Appendix D.

By using the coding standard, developers agree that they can understand other developer's code better without spending much time. One developer stated that, "I like a formal coding standard. It gives the same style and format, and this consistency helps me a lot in understanding other developer's code." However, one developer worried about putting too many coding standards on developer's shoulders. He stated, "Having coding standards is good, but too much coercion to the standards may hamper our performance because we need to look at coding documents back and forth while we are coding to conform to standards."

Table 26

*C# Coding Standard*

| Area | Sub Area | Description |
|---|---|---|
| Headers | File | The copyright notice and the CVS information are all that will appear at the top of a .cs file. These will account for the first 4 lines of each file. |
| | Class/Delegate/Interface | Class headers will be in xml format to facilitate the automatic documentation feature. If the class description is longer than one line, add a remarks section. |
| | Method/Event Handler | Function headers will be in xml format to facilitate the automatic documentation feature. If there are any exceptions thrown in the function, they must be documented inside the exception tag in the function folder. |
| | Properties | Class properties should be documented through 'summary' and 'value' tag. A remark tag may be used for more detailed explanations, but generally will not be found in properties. |
| Naming Conventions | Capitalization Rules | Class, Enum type, Event, Interface, and Method should use Pascal type. Parameter, Variables, and Protected should use camel type. |
| | Abbreviations | Do not use abbreviations or contractions as parts of identifier names. Do not use acronyms that are generally accepted in the computing field. |
| | Type Name Confusion | Different programming languages use different terms to identify the fundamental managed types. Class library designers must avoid using language-specific terminology. |

*Documentation.* After the firm adopted the Scrum process, the amount of documentation was reduced significantly. For example, detailed design documents including class diagrams (see Appendix F), sequencing diagrams (see Appendix G), activity diagrams, communication diagrams, and use cases were reduced or eliminated. A developer mentioned that, "Before we went agile, we required that detail design documents were prepared before code was written. But we are not creating detail design documents anymore." A QA manager also uttered that, "The detail design documents were routed to the QA manager, development manager, document manager, and sometimes to others." She also explained how detailed design documents were utilized:

> As the QA Manager, I read though those documents with a red pen in hand. I verified that we were standardizing things. If I had questions that were not addressed in the design about how something should work given a certain scenario, then I made sure that those questions were answered and the design document updated. The documentation manager also reviewed the document to make sure that the error messages, field prompts, reports, etc. were grammatically correct and that everything was spelled correctly.

A certain amount of documentation seems to be very useful when developers work on a complex project, try to find and turn around fixes and problems, or need some ideas and questions for the project. One project manager said:

> When we were working on a complex project where we would make one change and have QA test, then work on the next change, test was very successful. Documentation was there and notes were taken as we went. We were able to find and turn around fixes and problems quickly using a document. Documentation, the input of QA, and others who were familiar with the project, gave us a wider range of ideas and questions for the project.

Several subjects noticed that the reduced amount of documentation is a big headache. A project manager worries about quick implementations without creating detailed design documents. The quick implementations seem to cause ripple effects that

damage other parts of the project. She noted, “One drawback is that there are no detailed design documents. Whatever the idea is, it's just implemented. No one is taking the time to think about what effects it may have on others parts of the program.” Another problem is the increased number of bugs due to the lack of standardization of features, field names, and error messages. This kind of standardization would have been easily implemented if the documents had been created. A software tester stated, “There's no thought put into the standardization of features, field names, error messages, or anything. It appears that code is just thrown together. So, when I go to test it, it falls apart.” A QA manager confirmed this statement as she mentioned that, “I'm finding twice as many bugs since we went to agile as I did before we went agile. Once I have the basic bugs resolved, then I can really start testing it. And the system doesn't work the way that I anticipated.” The increase in bugs requires a lot of developer working hours to fix, and this is a major issue because the code should be re-written. The QA manager cited this example:

> I go to the programmer and they didn't even address the issue because it wasn't part of the design and they don't know how it should work. I end up calling a meeting with part of the team to discuss design issues that should have been addressed prior to the project even being coded. And most of these are not little design issues that a couple lines of code can fix. These are major issues and major code re-writes. And then once it's done, I've got to start testing everything again.

Several developers agreed that design and documentation should be a necessary part of completing a project. Developers are not in favor of reducing the amount of documents. A developer explicitly expressed his opinion when he said, “I don’t like the fact that our design and documentation requirements have lost their focus in Scrum.”

*Formal Code Review*. ABC has utilized formal code reviews since they started producing high-quality software applications. A web-based, formal-code review tool was created by a project manager in the firm and it has evolved to the current version through several revisions. When developers write or modify the code, they are required to go through a formal code review process before they check the code into the central code repository. Developers need to choose two code reviewers, and one of them is usually a senior software engineer or a developer who knows the area well. Once the code is delivered to the reviewers through the web portal, the reviewers provide feedback about if there is a side effect, a ripple effect to other code, or the code looks good. Developers and code reviewers sometimes have a meeting to discuss code changes that have a heavy impact on other modules. In this case, other senior developers or project managers are invited to the meeting.

All interviewees agreed that the formal code review was a necessary and essential step to construct robust applications. One developer declared, "I think the formal code review keeps us in the right track. We can find inaccurately written code in the initial development phase and fix mistakes overlooked by a developer." Another developer commented, "The formal code review can keep our code from being exposed to vulnerabilities such as buffer overflows and memory leaks. This process makes our code more robust and secure." A project manager expressed the following:

> We go through a very thorough code review. Often times, our code review requires a line by line review. We can avoid a lot of common mistakes by having two or more people examine the same code though it sometimes takes more effort and time. It has been proven that our code review process is effective at finding defects in the code under review. I think the formal code review is a vital and critical process in creating high-quality software applications.

*Environment Factor*

Four identified elements of the environmental factor include 1) customer involvement, 2) working environment, 3) interdependency among modules, and 4) the group of social facilitation, social loafing, group motivation, and evaluation appreciation. They are discussed in the following section.

*Customer Involvement.* One of characteristics of agile methods is a constant customer involvement in every aspect of the software development process. As part of the customer involvement, a customer representative is required to attend meetings. However, ABC could not invite customers to all of the meetings, such as the daily Scrum and the Sprint planning/review meetings, because there are so many customers scattered across the United State. Instead, product managers visit customers on site or talk to clients through WebEx to gather the project requirements, to show the progress of the products, and to obtain feedback from them. A project manager related:

> We spend almost a half of our work time in talking to our clients. Through the conversation with clients, we gather project specifications, show them the progress of our products, and receive their feedback. If possible, we use WebEx to show features of the products and some charts and graphs, otherwise, we visit them on site.

Another project manager spends many of hours visiting agencies and comes back with project requirements. He then creates a to-do list and a graphical user interfaces for developers. He mentioned that

> I go out to meet our clients and conduct research on what we need in a product. I come back to company with lots of requirements that our clients want. I produce a list of things we need to create and talk with developers to develop the product for our clients. I spent about 9 months last year doing research, going out to visit each agency. That is my job, just going out and gathering requirements, keeping track

> of equipments, putting together wire frames of what the screen should look like, what kind of data should be on it, and how it should be presented, and then take it back and review it.

ABC also hosts users' conference once a year to get feedback from the customers. At this conference, customers vote for or against a new direction of application development. A project manager articulated that, "We've been hosting users' conference for several years, and it has been very successful. One of the important events in this conference is that customers can vote for or against a policy and a direction of new application development."

One interesting notice associated with the customer involvement is that QA people have a hard time providing customers with a quick turn around on bug fixes because of Scrum settings. A QA manager stated that, "Going to agile has been a hard adjustment for most of our customers. We're unable to give them the quick turn around on bug fixes like they used to get. We can't give them a quick turn around on bug fixes because we can't interrupt the Sprint." She continued to explain it by saying that, "Before we adopted Scrum, we had been able to fix customer's bugs quickly. But now, we focus more on the code created during the Sprint and place less priority on customer's bugs because we don't want to interrupt the Sprint process."

*Working Environment.* An open working environment is recommended by the Scrum method because it promotes communications, facilitates self-organization, and helps developers get together easily. Before ABC started the Scrum process, most developers had their own office. After adopting the Scrum process, all the developers were reorganized into cubicles in such a way that all the team members are placed inside

the same cubicle area. Most developers do not like this new work environment and consider the cubicle settings less efficient because they cannot concentrate on their work while their coworkers talk to other coworkers. One developer complained:

> We share cubicles with a coworker. I think this is a bad thing, because we are constantly distracted by the person we work with. Cubicle-partners are often having conversations with other coworkers about their tasks, and it is hard not to get drawn into their conversations, even if you don't have anything pertinent to add to it. Overall, I feel it is distracting and generally less efficient.

Another developer stated, "Don't pair developers up in a cubicle with someone else. Place them near team members, but give them enough space that they can concentrate when necessary." A QA person also mentioned, "I was more productive when I had my own office. I was not distracted by my coworker's phone conversation or other noise going around me."

However, some developers think that cubicle settings increase the amount of communication between team members. One developer mentioned, "I like the open-space-working environment. It promotes our communication. I can easily grab one of my team members and discuss issues and problems." Another developer stated, "I feel I can talk to our team members easily and ask questions quickly." One developer proclaimed, "I like the idea of placing every team member inside one cubicle area because I think it fosters collaboration and teamwork."

*Interdependency among Modules.* In the first stage of the projects, the Scrum method was not very effective in terms of bug rate. During the first three months, for example, the QA department found almost twice as many bugs after the firm switched to the Scrum method. A QA manager confirms, "I'm finding twice as many bugs since we

went to agile as I did before we went agile." They attributed the finding to the complexity of the project. As the size and complexity of the application grew, the dependencies and interconnections among tasks in the application increased. However, the developers were not able to fully consider all the dependencies and interconnections among modules because of their narrow-minded planning and design in each Sprint planning meeting. A developer explained:

> Team members seem not to take enough time to think about interdependencies and interconnections between modules. After several Sprint cycles, we realized that we didn't think about what effects this module may have on other part of the project or how this module may be interrelated with other tasks in the product backlog.

It was also noticed that when developers had to complete a set of tasks, determined in each Sprint planning meeting, they had a tendency to complete tasks in a quick and dirty way rather than to think of how the code will be maintained, and how each task in the current Sprint planning meeting can be flexible for future changes. A developer stated, "We have a tendency to do things in a quick and dirty way rather than to think into the future. (How will we maintain the code, and will it be flexible enough for future needs?)" Another developer noted, "Sometimes things are rushed into the Sprint meeting and the team doesn't take the time to add such tasks into the Sprint backlog." Further, the developers often found that their monthly work schedules for producing intermediate deliverables were optimistically estimated, failing to finishing all the tasks in the Sprint backlog. A developer mentioned that, "When we plan things in our monthly Sprint planning session, we underestimate how long things will take, and that leads to the failure of finishing all the tasks in Sprint backlog." A project manager also

stated that, “Some tasks just keep getting carried over to the next Sprint backlog. They hope to do better the next month in estimating their project.”

*Social Facilitation, Social Loafing, Group Motivation, and Evaluation Apprehension.* This section is quite different from the other sections because it was developed through observations based on social facilitation, social loafing, and group motivation. Social facilitation indicates that in the presence of other people, performance is focused on a simple task, whereas performance on a difficult task is hampered (Aiello & Douthitt, 2001). Social loafing is the tendency to take advantage of others’ efforts when working in groups, while group motivational gain is obtained when people increase their effort to help co-workers whose performance is poor.

Social facilitation effect and group motivational gains were observed as factors that reduced development times and lowered bug rate. In particular, evaluation apprehension (Bond, 1982; Cottrell, 1972) was observed when the firm invited all developers, QA personnel, and people in other departments (sales, marketing, customer support) to the Sprint review meeting. In the Sprint review meeting, all developers had a chance to show what they had done in an interdepartmental environment and they were concerned about how they were evaluated by these people. Group motivational gains were also obtained as a social compensation effect (Williams, Harkins & Karau, 1991) when a team member helped other team members who were not familiar with new development tools or programming languages. At the same time, ABC did not have ways to evaluate individual performance, leaving room for individuals seeking to free-ride.

*Information Systems and Technology Factor*

The information systems and technology factor is comprised of communication system, information and knowledge sharing system, and bug tracking system and management tool. These concepts are discussed in the following section.

*Communication System.* It seems ABC realized and understood the important role of communication in the software development process because they have put a lot of efforts into establishing good communication channels. As mentioned in the third chapter, ABC had two development sites for about a year and developers were divided into two geographic locations. With the two development sites, they set up devices to have video conference capabilities between the two sites. However, due to the time and use constraint, the video conference was not always available for each individual software developer. Rather, the video conference was usually used for team meetings between sites once or twice a month. Because of the unavailability of video conferencing, phone conferencing was used a lot.

> It seems that phone conferencing has problems as well. A lead engineer stated:
>
> > When we have the daily Scrum meeting in the morning we did it in a couple of different ways. One was video conferencing and that was actually quite good. We also did phone conferences, which were not as good as video conferencing, mostly because when somebody made a comment, you couldn't see their face, which gives the background. Of course, the Scrum master couldn't do anything about that. That was counter-productive. If you have two groups video conferencing, that is much better. You can see face-to-face and if somebody is upset, you know about it. You can't hear people making faces over the phone. That did happen.

Other developers also mentioned in the survey that video conferencing helped them to see the other person's facial expressions, gestures, and body language. One developer

stated that, “it’s still hard to know whether they understand what I am explaining to them”.

Though video conferencing is much better than phone conferencing, there are still limitations to using video conferencing. The daily Scrum meeting and the daily Scrum of Scrums meeting have mostly been done through phone conferencing. The monthly Sprint planning and Review meeting have been done through video conferencing. Other than video and phone conferencing, software developers have been using a tool called web demo (http://www.beamyourscreen.com) to show people in remote sites things on someone’s screen. This tool, together with a phone system, seems to work very well, as a developer mentioned,

> Actually we used web-demo which worked out very well for us when we are going to do code reviews and we used that web demo with phone conferencing. And it was like being in a media-room. Actually, it was much better than being in a media-room in a lot of cases because we were able to scroll to where we want to look at. It was very efficient.

The last multi-media that the company has been using are an Instant Message (IM) and an email system. All of the developers mentioned that they were using the instant message and email system much more than before they were divided into two groups. It seems the instant message system is used more effectively than the email system for a short and quick question and answer. The instant message system was also used more by new employees. Several developers agreed on the following statement:

> Yea, a lot more when you are working with a new person. You are using it (IM tool) all the time because he’s asking questions back and forth over an IM session. You are using it a lot with a new person when, if he would have been here, he would have come in the office to show to him instead email him back and forth, or you would have been looking at it together. You could have used web demos to do that but in most cases it is short questions. We didn’t feel like we had to do something like that.

One developer, who used to use VPN mentioned, “Sometimes, the VPN is too slow, which causes frustration.” It was obvious that people did not use VPN once they found it was not fast enough. Though the video/phone system, web-demo system, and instant message/email system help software developers reduce the geographic distance between two sites, it seems those multimedia systems have not been working as optimally as face-to-face conversation. Several developers mentioned that, “It would so much easier to be able to work with somebody right here in my office.” An executive officer also mentioned, “Well, in some ways it would be better to be down there at times.” He went on to explain that

> it hasn’t been that big of a disadvantage but once in a while, we just get two or three guys together, if there’s an issue and somebody wants to talk about options or kind of brainstorm on the idea. For me to join that group from this remote site is more difficult.

The company seemed to suffer a lot when a communication line was down because of their heavy dependence on it. A lead engineer mentioned that, “We had a few problems last month with phone systems not working very well. When the phone systems were down it was a big deal, plus when phone systems were down the data line went down too.” Because the telephone systems and data systems share the same physical line, employees could not do anything, as a project manager stated:

> For example, you’re working on a summarizer or something like that, all of sudden, the screen is closed down. And if you are in the middle of some debugger and the data line is gone, you will lose everything and you have to start it from the beginning. That’s why one of our programmers went over home. She’s working out of her home now.

*Information and Knowledge Sharing System.* When the main part of ABC moved to a different location, the firm went through major changes. One of them was a change

in staff. The vice president of the software development division related that, “There was a change when we moved to a different location with new staff, and I think we lost about 15% of our staff, including software developers.” To fill the empty positions, ABC hired several new software developers. These newly hired people created more bugs; a project manager said, “There are a lot more bugs now because people are not always knowledgeable about the software that they are working on” and “often they change some code, something like a radio log which is very complex piece of code and slight changes to that have a lot of impacts in lots of areas. It takes a time to be familiar with our complex systems.”

Because of the complexity of ABC’s software system, it is important to have well-structured information- and knowledge-sharing systems between experienced software developers and brand new software developers. In particular, it is critical to have such a knowledge sharing system if new software developers in one location need some of the expertise of software developers in another location. One developer stated, “It is a little difficult to share information and explain ideas over those large distances.” Another way to get around this problem might be placing new developers with senior developers within the same Scrum team in one location. But this solution is not always feasible for ABC because of the complexity of software systems and the situation that developers often need expertise from other developers in a different Scrum team and in a different location. So, to facilitate the knowledge and information-sharing, the firm has been using a web-based Wiki program (http://www.wiki.org) that enables developers to add and edit items that might be critical to other developers. For example, the section called “gotcha” in the company’s Wiki includes the most frequent mistakes that

developers can make in many different parts of the company's software system. Due to easy access to the Wiki program, the information stored on the Wiki database mitigated the problems between the two sites.

*Bug Tracking System and Management Tool.* ABC has been using a bug tracking system called MOM, which was developed by the firm on the UNIX platform. The MOM bug tracking system is shared by many groups, including support people at the help desk, developers, and QA personnel. A brief description about a bug and the step-by-step procedures to duplicate the bug are entered into the system. One developer mentioned that

> we use our own bug tracking system called MOM. QA personnel and support people usually describe the nature of a bug, the place where it is found, and the procedures to duplicate the bug if possible. Once the description is entered into the system, it can be viewed by developers, QA personnel, support people, and managers.

Based on the level of severity, each bug is labeled with a number between zero and three. The number zero represents very urgent and the highest level of severity, whereas the number three represents the lowest level of severity. A person who is assigned to a zero level bug is required to stop all the current tasks and work full-time on the bug until it is resolved. One developer mentioned:

> All bugs are entered into MOM and are assigned to an appropriate developer or group of developers. A zero level bug is uncommon but when it happens it is very critical to solve the problem as soon as possible using all possible resources because the zero level bugs could stop some parts of or the entire operation.

As mentioned earlier, the firm adopted a commercial bug tracking system called JIRA (http://www.atlassian.com/software/jira/). A vice president in the software development department stated:

> Well, recently we moved away from the MOM system. We used to create support problems and then create a duplicate for development in the MOM support system. We have now moved to JIRA. I think in MOM you didn't have fields for certain information. Everything was pretty much put into a narrative environment or a report.

JIRA is used to record defects found in alpha and beta testing and in other areas. A manager stated that, "JIRA only logs defects that we found in alpha and beta testing and also defects in standard code that a customer told us about. It's just a tracking tool for the development division for defects in products." It seems JIRA has filter functions that MOM does not provide. The manager mentioned that, "JIRA has a lot of information that used to be found in MOM. But it has important filter functions which we couldn't do in MOM. It has some advantages so that I don't want to go back to MOM." Another advantage of using JIRA seems to be that it provides priority codes that rank the level of the severity of bugs. This function automates the selection of a next bug to be tackled by listing the bugs in sequential order based on the priority. The VP in software development stated:

> Well, one of benefits of JIRA is that it has severity priority codes. It was a part of BETA test, and there's a number of other fields that down the road we will be using to divide the information. We are working a way to try to determine as efficiently as possible what bugs to work on next. JIRA can still automate the choice versus having a person just look at and say what you need to fix or not. We try to use that information to do that.

One of disadvantages of using JIRA appears to be that it does not offer a decent search function on the bug information already typed into the system. This is mainly caused by the JIRA interface, which requires a user to enter the narrative bug information into separate text boxes.

In order to further improve the effectiveness of project management, ABC used two different managing tools: Microsoft Excel and VersionOne. Microsoft Excel was used first, but was replaced by VersionOne because of Excel's limited capability for efficient project management. VersionOne, as mentioned earlier, is a web-based commercial management tool that provided various functions, such as simplifying project planning and management, enhancing business and project adaptability, improving project visibility, and increasing project predictability and confidence. Some developers expressed difficulties in using three different bug tracking and management tools. One developer complained that

> . . . it's hard because we are using MOM for part of our tasks, we are using JIRA for another portion, and we are using VersionOne to track the development cycle and Scrum cycle. So, it makes difficult that way. We can't use one product for everything.

### *XYZ Firm*

The XYZ firm had five Scrum teams, and each team consisted of three or four developers, a project owner, and a Scrum master. Due to the small size of the firm, only one quality assurance person, and one database administrator were designated to provide services for all Scrum teams. Every Scrum team worked on a new project except for a team that worked on on-going maintenance. Table 27 summarizes the common categories, concepts, and data related to XYZ. As shown in the table, the factors of human resource management, structured development process, environmental, and information systems and technology were constructed for XYZ. The concepts comprising these categories are discussed in the following section.

*Human Resource Management Factor*

The concepts of training, collaboration, and multiple responsibilities constitute human resource management factor. Each concept is discussed in turn.

*Training.* It seems that XYZ needs to provide developers with more formal step-by-step training. Most developers feel that they did not have enough training on the Scrum development method. Due to the lack of training, some of developers do not see the big picture and the benefits of Scrum as a whole. One developer mentioned, “I don’t think our whole team buys-in to Scrum development in general because we are not getting a whole picture of how Scrum is working and how it benefits us.” Another developer confessed that he had a misconception on the relationship between a project and a Sprint. He thought he should complete a full project within a single Sprint. In reality, a project is usually completed through multiple Sprints.

Table 27

*Categories, Concepts, and Data Related to the XYZ Firm*

| No. | Common Categories | Concepts | Data |
|---|---|---|---|
| 1 | Human Resource Management Factor | Training | • Developers need more formal step-by-step training<br>• Some of developers do not see the benefits of Scrum<br>• Company provides brown back lunch training and a "luncheon learn" meeting |
| | | Collaboration | • Scrum provides good collaboration mechanism between developers<br>• Collaboration between developers and product managers should be improved<br>• Scrum helps developers to be aware of other developers' tasks. |
| | | Multiple Responsibilities | • One person is responsible for many tasks in different field<br>• Product manager is a bottleneck in the development process<br>• Developers have a communication problem with a product owner |
| 2 | Structured Development Processing Factor | Scrum Framework | • Scrum ceremonies force developers to be on the same page<br>• Daily Scrum meeting, Sprint planning meeting, and Sprint review meetings are sometimes inefficient<br>• Setting up the meeting time is difficult |
| | | Formal Code Review | • Company employs informal code review<br>• The formal code review can impose accountability to developers |
| | | Unit and Integration Testing | • Company uses a N-unit testing<br>• Testing self-created code is not efficient<br>• Legacy code is not suitable for unit or integration testing<br>• Wide range of testing skills are needed |

(table continues)

| | | | |
|---|---|---|---|
| | | Coding Standard | • The coding standard can provide easy maintainable code<br>• Company has a verbal coding standard<br>• The coding standard can hamper developer's creativity and reduce the efficiency |
| | | Documentation | • Less documentation<br>• More comments on the code<br>• Hard to complete the system without having any documents<br>• Equally shared skills and knowledge among team members |
| | | Project Estimation and Planning Poker | • Developers have difficulties to estimate legacy code related project<br>• Planning poker reduces estimation difficulties<br>• Breaking big tasks into smaller ones helps developer have good estimation |
| | | Use Cases | • Developers realize the importance of creating use cases<br>• Use cases help developers understand the system they are going to build<br>• Users do not know what user cases are |
| 3 | Environmental Factor | Customer Involvement | • Not involved in the decision making process<br>• Biggest roadblock in the development process<br>• Customers do not know what they really want<br>• Customers give unclear system requirements |
| | | Working Environment | • Open working space makes developers to be easily accessible<br>• Open working space fosters communication<br>• Open working space distracts developers |
| | | Common Tools and Problems between Teams | • Similar technologies can be used between teams<br>• Common problems between teams can be resolved through collaboration<br>• The role of liaison between teams is needed |

(table continues)

| | | | |
|---|---|---|---|
| | | Government Project and Scrum Method | • Government project requires a big planning and a big design up front<br>• Developers work with many unfamiliar jargons and acronyms<br>• Company needs a new hybrid development method |
| | | Social Loafing | • Everyone is fairly motivated and quick to point out if anyone is not doing his share<br>• Individual hard work is not recognized<br>• Accurate measurement of individual performance is needed |
| 4 | Information Systems and Technology Factor | Communication | • Daily scrum meetings improve communication within a team<br>• Lack of communication between teams<br>• Work is being duplicated<br>• Lack of communication with customers |
| | | Bug Tracking System | • A web-based bug tracking system is helpful in prioritizing bugs and keeping track of bugs.<br>• Mantis, a free bug tracking system works well except for some cases<br>• Developers want customers to report their bugs in Mantis |
| | | Version Control Systems | • Open source revision control system called Subversion is a main system<br>• Subversion provides a decent tagging and branching method<br>• Subversion rebuilds the system automatically if code changes |

When developers first began to work on Scrum projects, their project manager demonstrated the concepts of Scrum projects and how they related to the existing systems. After that, developers were to learn the Scrum concepts and working environments all by themselves. One developer commented, “We are usually taught briefly by a project manager how the system works the first time and then kind of go from there, I guess.” Another developer stated, “I did not have formal training. I looked up Scrum in the Wiki to get an idea what the working environment looked like.” Many developers want to have a continuous formal set of training. One developer noted that, “It would be helpful if we have a class that trains us, more than just an overview or just a kind of tour around.”

To mitigate problems caused by the lack of a formal training program, the company recently began hosting regular brown-bag lunch meetings to provide more training. A Scrum Master affirmed that, “Recently, we have been hosting more formal lessons on how things should work through a brown-bag lunch. I have been attending BB lunch regularly. It’s really helpful. Other than that, it’s really been just a brief overview.” In addition to the BB lunch meeting, the company offers a “lunch-and-learn” program. An operations director explained, “We do have pretty frequent lunch-and-learns where the company buys us lunch and then we have someone like a operation director go through and explain how Scrum works.” One advantage of having the lunch-and-learn program is that everyone including developers, Scrum Masters, Product Owners, Project Managers, and other staff members can attend the meeting and provide feedback. The operations director said, “The whole company can attend, not just some people, so that everyone can provide feedback.  It also helps people to be reminded how to do tasks as

well, because sometimes through the day-to-day we forget certain aspects of Scrum, so it's helpful to be reminded of them and remember to do it right."

*Collaboration.* It appears that the Scrum method has provided the company with a good collaboration mechanism. Most developers feel that they have good collaboration between developers within the same team. One developer mentioned that, "I think it's really good. I have often seen a team member go to another to ask about coding practices and methodology. I have seen that several times so I think that's really good." Another developer stated:

> Whenever I have a problem or issue to discuss with other developers, I am very comfortable talking to any of my team members. I think Scrum practices promote collaboration between team members. For example, throughout the daily Scrum meeting, we can identify what roadblocks we have to complete tasks specified in a Sprint backlog and what items need to be done together.

Further, it seems the Scrum method has helped developers be aware of other developers' tasks and be interested in other developers' success. This atmosphere seems to make developers self-motivated. One developer stated:

> It seems everyone is involved and they are interested in everyone's success. Everyone is very self motivated. So, if they don't have something to do, they go in and look at our Sprint backlog and choose the next thing. So, it's more reach out and get than assigned to us.

However, some developers think the collaboration needs to be improved between developers and the product manager. One developer stated that,

> "There has been great collaboration between developers, but there has not been enough collaboration between developers and the product manager. He has been busy with many tasks and has not been able to allocate enough time to discuss problems and issues with developers."

The collaboration issue between developers and a particular product manager is elaborated in the multiple responsibilities section.

*Multiple Responsibilities.* The Scrum method does not recommend that developers be involved in multiple projects at the same time. At XYZ, all developers work on a single project. The one person with multiple responsibilities acts as the product manager, product owner, and accounting manager. Due to his multiple responsibilities, he is too busy and overloaded. Developers have a tendency not to talk with him enough outside of Scrum meetings or ask him to deal with other issues when they come up. One developer stated, "He tends to be a bottleneck for the rest of team members. We've been waiting on him giving us feedback. Sometimes it tends to be a little bit of lag in getting any questions answered." Another developer mentioned:

> You know we have a product owner, but he is overworked and he has too many other responsibilities so he can't dedicate himself to the product and can't get all kinds of tasks done. It hinders our progress, and he doesn't have enough time to find what exactly we have to do. So, we need to get someone else to help with that.

Developers have also a communication problem with him. One developer articulated:

> The only sort of communication problem we have is with the product owner. He is overloaded with too many other tasks throughout the company, so he is rarely available, and we have a lack of communication with him. So, that hurt us, but when he is available, the communication is fine.

It seems the company needs to release some of this person's responsibilities so he can concentrate on a single project and make himself available to other developers for more decent communications.

*Structured Development Processing Factor*

As shown in Table 27, structured development process factor consists of Scrum framework, formal code review, unit and integration testing, coding standard, documentation, project estimation and planning poker, and use cases. Each concept is presented in the following section.

*Scrum Framework.* It seems that Scrum has been well adopted and has brought big improvement to XYZ. One developer stated, “Scrum agile has been working quite well for us on all of our projects. It has been a huge improvement over the waterfall method.” Another developer mentioned, “Scrum really helps team members to get involved in the project.” One project manager also noted, “The Scrum method helps team members to be aware of everyone’s progress, and in Scrum, nobody can fall behind without people taking notice.” Another project manager mentioned, “The Scrum framework prevents a scope creep because tasks can only be dropped from a Sprint, and Scrum also prevents the project from going too far off course if client’s requirements are not accurate.”

Scrum ceremonies including the Daily Scrum Meeting, the Sprint Planning Meeting, and the Sprint Review Meeting, seemed to help developers focus on producing quality applications. Most developers testified that the Scrum ceremonies have been very useful and very productive. One developer mentioned that “The 15-minute standup Daily Scrum meeting has allowed us to be on the same page because we can talk to each other and everybody knows what other members are working on.” Another developer expressed a similar opinion on the Daily Scrum meeting. He mentioned:

> This is a great way to make sure everyone is on the same page regarding work accomplished and work to be done. This allows quick assessment of those people

> who are already behind from the beginning and forces the team to make an adjustment and compensate.

A project manager noted that, “The daily Scrum meeting is valuable and essential. We try to include clients in these meetings as much as possible.”

Regarding the Sprint planning meeting, most developers considered it as an imperative and critical meeting. One developer noted:

> Time spent in elaboration during the Sprint planning meeting significantly helps estimating and scope planning because we are able to determine the number of employee working hours for the next 30 days and how much work we will be able to accomplish. Additionally, we have 3 or more developers estimate each development task which promotes in-depth planning discussions when estimates vary widely. This creates a more accurately defined Sprint backlog.

One developer regarded the Sprint review meeting as a bridge meeting which leaded people into the next Sprint. He stated that, “The Sprint review meeting is important for transitioning into the next Sprint. Depending on the project, this may be a good time to plan a short (around 5 days) Sprint to develop post-launch patches that may be critical for the client.” Another developer also mentioned that, “This is very advantageous as long as all team members, including the client, are open-minded enough to have a candid discussion. It allows all team members to discuss the items that did not work well and what can be done to address them in future Sprints.”

Regarding a product backlog and Sprint backlog, one developer stated:

> The master product backlog is invaluable in long-term planning as well as quickly ascertaining the next tasks. With our clients, we hold a master product backlog reorganizing meeting every few months to make sure the priorities are still specified accurately. The Sprint backlog is the key to knowing what everyone is working on, how much time is left, and what tasks are left to do. Keeping the Sprint backlog in priority order and working on tasks in that order assures that the most important stuff is done first.

Another developer also mentioned:

> Product backlog is important, especially for a project that is going to span multiple Sprints. It will help keep track of remaining tasks, help you prioritize them, and prevent them from falling through the cracks between Sprints. Be sure to update it as new feature requests come in. Having a product backlog makes Sprint planning much easier. Sprint backlog is the output of your Sprint planning meeting, and an essential tool for the daily Scrum.

However, some developers talked about inefficient Sprint planning and review meetings. One developer argued that, “Some of our Sprint meetings are so simple and it seems to be a waste of time spending a whole day just for planning and review. I think it needs to be adjusted based on the complexity of the project that we are working on.” One senior developer noted that, “Keeping daily Scrum meetings to 15 minutes was difficult. Some of this is caused by just gabbing a little too long, but valid reasons for taking too long are the amount of material we needed to discuss, and also because we hold Scrums for 2 or 3 projects at once.” Another developer mentioned:

> Our daily standup Scrum meetings sometimes go on a little longer just because everybody is talking about what they did last night. I think there probably is some good advice on trying to keep your daily standup meetings consistent and short so that people are not distracted and they can go back to work quickly as most people would rather work productively than waste time.

Another issue is related to setting up the meeting time. Due to the flexible work schedule among developers, it is difficult to get all developers together at one time without interrupting their work. One developer stated:

> I think the hard things for us in Scrum is when to do it because some of us get in at 7:30 am and some of us at 9:30 am. So, as a team, we just have the Scrum as soon as everyone gets in. That’s usually at ten or eleven. The problem is that those who get in early are interrupted from their work because they’ve been working very well for two or three hours. They are in the grove or zone so being interrupted is frustrating. We talked about doing it at the end of the day but that also has a problem because some people come in at 6:30 am and leave at 3:30 pm,

> and some people come in at 9:30 am and leave at 6:30 pm. It makes it hard for our team to get together all at one time.

Some of developers feel that Scrum might not be appropriate for large-scale projects because it is not easy to make a large team be agile. One developer stated:

> Agile is difficult with really large projects. It's hard for large teams to be agile. It is necessary to split the large project up into teams of 8 with common goals. Let team of 8 Scrum self-manage. Project Managers can facilitate communication, and check status between teams.

Another developer made a suggestion about tasks on the Sprint backlog. He suggested that tasks on the backlog should be updated dynamically and that tasks taking more than a day should be divided into small sub-tasks. He suggested:

> When updating the Sprint backlog every day, if the tasks you are working on don't match the granularity of the tasks on the backlog, update the tasks on the backlog and redistribute the hours among the tasks. This is to prevent treating a group of tasks as a single bucket of hours that you are working against, which makes it more difficult to see exactly where the progress is. If a task is more than a day of work, try to break it up into sub-tasks.

A developer noticed that some of Scrum team members were not flexible and did not spend enough time to create a detailed Sprint backlog. He stated, "I would like to suggest that people participating in Scrum be flexible. Rigidity seems to be the bane of Agile. Also, when we generate a detailed Sprint backlog we need to spend as much time as possible." One last comment from a developer showed that some managers tried to manage the Scrum teams instead of coaching them. He asserted, "Managers – avoid managing! Let the team self-manage. Managers/Project managers should act as coach only – it doesn't work if the project manager is running the Scrum like a status meeting. Project manager – step back and let the team run the Scrum meeting."

*Formal Code Review*. Developers have not set aside a time for a formal code review, but they have been having informal code reviews. Due to the informal code reviews, tools that facilitate formal code review have not been developed and utilized. In addition, the informal code reviews has not been employed frequently. One developer stated that, “We don’t incorporate proper tools in our code review.” Another developer mentioned that, “We have an informal code review occasionally. We don’t do it very often and probably should do it more. We talk about it a lot but just never have the time to do it.” It seems developers are aware of the benefits of having a formal code review though they do not use it. One developer stated that, “Other people can offer feedback on how a developer is doing and how the particular code can fit into the rest of a project.” Another developer also mentioned that, “Through a formal code review, other people in the team can look at the assumptions and choices that I made and say this is a good work for that situation or not good for the situation.” Another benefit of having the formal code review is to impose accountability on developers because they know their code will be reviewed by other people. One developer stated:

> I think it’s really useful. It gives you accountability because you know at some point somebody else will go back and look at your code. In the short term, developers may not pay extra attention to their code because a lot of developers just take an assignment and write the code with the belief that no one will look at it again. Formal code reviews would solve this problem.

Most developers agreed that visiting the code retrospectively is important in terms of improving the quality of code and enhancing developer’s coding skills.

The Scrum master mentioned a problem that might be caused by the lack of formal code reviews. He stated that, “There is a chance that a developer writes code which is already built in, for example, in the Dot Net framework or ASP Dot Net

framework." If a developer tries to write code which is already written, tested, and proved to be efficient by a third party, it would be a big waste of time and money. It appears that developers are good at looking at other developer's code and giving good feedback. It seems it is just a matter of setting a time aside for a formal code review and selecting an appropriate tool to facilitate the code review.

*Unit and Integration Testing.* The company has been using a unit testing tool called "N-Unit," which provides a unit-testing framework for all Dot Net languages. Each developer tests their own code, but they try not to because they think testing their own code is not effective. One developer mentioned that, "We test our own code but we try not to because it's not effective to test one's own code, and developers usually have an assumption that their code always works." When developers test their own code they also consider an integration test. One developers stated, "Everybody tests their own stuff just to make sure it works and also making sure it works with the big picture and everything else works with it." It appears that several quality assurance personnel can help other developers in testing regardless of team boundaries. A quality assurance person stated that, "We have a few people who do help out with testing. They are technically on the team but they are just available for everybody to help testing." In addition to internal testing done by in-house people, clients are also involved in testing by visiting a test site and conducting a test. A project manager stated that, "Generally, when the application is getting ready for clients, the clients get into a test site and test. They do track and test, and enter bugs that they find."

It appears that the firm has a large legacy code base and the legacy code was not designed for unit or integration testing. This is a big challenge for the firm. A project manager stated that, “We’ve got all legacy code and it wasn’t written really for test cases. It would be nice if we can figure out someway of going back and kind of cleaning some of those things up. In terms of quality assurance, that’s really about biggest challenge right now.” The firm also has code, called “code behind,” which is difficult for developers to test because that code works behind the curtain and there is no efficient way to test the code.

Developers wish to have a wider range of testers. One developer stated:

> Probably, we would get managed better if we had more people who have a wider range of software testing skills. Right now, usually, one person is testing, sometimes someone who is technical and sometimes someone who is non-technical. They usually catch different types of bugs.

Another developer uttered that developers could not test everything because clients did not have enough budget to cover one hundred percent testing. He also explained the process of a unit and product owner level testing.

> Our developers create their own unique tests. But we don’t have one hundred-percent coverage because our client doesn’t have the budget to make us test everything. We do create a unit test, then we have our product owner-level testing, then developers test their own code as well, and then from there we put everything in to MANTIS, which is our bug tracking system.

*Coding Standard.* Several coding standards for different computer programming languages exist. For example, Sun Microsystems provides the standard conventions for the programming language and Microsoft proposes coding guidelines for the C++ and C# programming languages. In addition to Sun Microsystems and Microsoft, other organizations such as GNU Free Software Foundation and CERT at Carnegie Mellon

University also suggest standard coding styles and guidelines. The main benefits of having coding standards is that developers can provide easily maintainable code, and that developers can decipher other people's code without difficulties.

It seems developers are aware of the main benefit of following the coding standard even though they do not have a formal coding standard. One developer stated, "There is no formal coding standard; probably a coding standard could be of good benefit to us." Another developer mentioned that, "There is no written coding standard, but we are close enough to propose to a person who is not following a norm or our verbal coding standard." Some of the developers employ coding standards provided by the tools they are using or modify suggested guidelines to suit their environment. One developer stated that, "The coding standard is dictated pretty much by MS Visual Studio." Another developer mentioned that, "I took the Borland coding standard and changed it and updated it for what I thought it would be best for us."

All interviewed developers think that they are doing fine without having a coding standard. One developer stated, "Most people have their coding styles, which are decent enough you can follow them pretty well. So, it's not a big deal." On the contrary, one project manager thinks that forcing the coding standard can hamper developer's creativity and reduce their efficiency. He stated that, "There are definitely cons to different styles of code being everywhere. People are creative where they can work the most efficiently. The problem with enforcing a coding standard is that it limits creativity." Another project manager mentioned that, "I think it is not worth to it force coding standards that hinder our performance if a lot of people have to relearn how to code in a lot of places.

*Documentation*. The Scrum method, like other agile software development methods, significantly reduces the amount of documentation. In fact, the agile methods claim that the code itself should be a document. That is why developers who are accustomed to agile methods place more comments in the code. Several developers mentioned they placed more explanations in any tricky piece of code and for any changes that they made. However, many developers agree that without having any documents, it is very difficult to complete tasks for those developers who are working on parts of the system they never worked on before and also for new developers who do not have much experience with the project. For both cases, developers who do not understand the project ask a lot of questions, which takes time away from developers who do understand the project. One developer mentioned that, "When I first got here, of course, I was overwhelmed. It would have been nice to have some documents that explain why certain things were done in a particular way and what they were." One more developer mentioned that, "Agile methods do not use specification documents. I think that might be a weakness in agile methods. The agile methods allow you to go much quicker as long as whoever is specifying has a very good idea of what clients want. If this is not the case, the agile methods are just as slow as anything else because you are going to have to get clarification."

Another developer also raised the issue of the lack of documentation. He stated "Right now, we have one guy who is the main guy. He knows all of the systems and I think, personally, that might be a mistake. Not because he is not good at it, but because it just makes one gigantic point of failure if he is hit by a bus or if he leaves for another company." It would take several months for the firm to recover the knowledge that one

main developer has. The idea behind reducing documents in the agile methods is to keep every team member equal by sharing skills and knowledge on the systems. In that way, if one person leaves, there is still a lot of shared knowledge that has gone around among other team members, so it is not a big deal. However, in reality at XYZ, this is not feasible.

*Project Estimation and Planning Poker.* The developers at XYZ had a hard time estimating how long a particular project should take. The level of project estimation difficulty increases when developers need to deal with legacy code or when developers do not have the experience required to finish a project. One developer said, “It’s hard to get estimation on legacy code.” Another developer stated, “We are just doing the first project and the accuracy of estimation is hard to determine because we just don’t have much experiences on this kind of project.”

To resolve or reduce the difficulties, developers were introduced to a new project estimation method called “Planning Poker.” In Planning Poker, all developers are required to pick a card, each of which has an estimation number to express their estimate of a particular task. If there is a big gap among developers’ estimation, developers discuss the task and try to narrow down the gap. One developer explained the procedure for Planning Poker as:

> For each task, we discuss for a minute or two and call the vote from the planning porker card. Everyone holds up their estimates and then if they are pretty close then we just take an average of them and put them as an estimate. If they are very far apart, if someone says two hours and someone says 20 hours then we pause and re-discuss it to find out why the estimates are so different. As soon as everyone has a thorough understanding of the problem, then we go again until we get closer.

Another developer commented:

> I guess the idea is to kind of narrow down the numbers. In the past we just made an educated guess, which is O.K. All the people on the product made a guess. There are usually three guesses, which result in three estimates on a task. If they are too different, then we talk about the task and figure out why we have different estimates. In Planning Porker, I guess there are ranges of numbers. I think 1, 2, 3, 5, 8, and then 13, 20, 40, and 100. So, when someone chooses 40 and someone else chooses 8, then we need to discuss about the task. If we are not sure about the task, then we talk about it and do something on the whiteboard to lay it out, and discuss what could be done.

Developers think that there are many merits to using Planning Poker. One developer stated, "You can get the opportunity to say why you think it's going to take so long, why you think it's going to take not nearly as long, and then you can actually work out some of the issues." Another developer mentioned:

> I think the advantage is that we can have a lot of discussion. That's an advantage. A few times I come through…because I don't know the whole systems as well as they do. I give some number and they give some other number. I think it eventually gets close together, but at least a project manager can tell me an overview about the things that are involved and make sure we are on the same page.

A project manager also stated:

> It is kind of nice to just have some sort of anonymous way we can throw cards up and discuss it rather than writing it down and be coerced into thinking by something or by whatever everyone else thinking. So it is nice…it helps us to have each developer have their own opinion.

It is noticed that breaking big tasks into smaller ones helps developers have better estimates. One developer commented:

> I think one of the things that really help is that everyone here seems to be conditioned to breaking their tasks down into the smallest possible measurable tasks and then to say, this field needs to be changed to accept this variable, or something like that. Very small tasks…you could say that yeah that task by itself will take me an hour or two. And so it's a lot easier to make the good estimates that way if you break it down into a small element.

Another developer also expressed the importance of breaking tasks into small one. He said, “We already have our tasks from our product backlog. We break them up into small ones first in the Sprint backlog so that we can have more accurate estimates.”

It is noticed that that every developer thinks Planning Poker is very useful, effective, and pretty reasonable as well because it gives developers a chance to express their honest estimation. However, experience, pre-knowledge, and skills are noted to be important factors in good estimation. One developer stated, “I think Planning Poker provides a more accurate estimation. That’s part of it. But I really think it’s just matter of experience for good estimation.” He also mentioned that, “I think it’s a good thing to have Planning Poker, even if it brings big differences in estimation numbers the first time around.” Another developer reiterated:

> It works well in the middle to the end of a project because now you have a very good idea of what you’re working with and the technology you’re working with. But initially, it got to be horrendous unless you’re working with technology that you’re extremely familiar with. Also, you know the technology is only half the battle. It’s the logic and the system itself that you have to figure out and sometimes it takes a lot more time then you think.

This developer argued that, though Planning Poker can help your estimation, the bottom line is you need to be familiar with the technologies that you are going to use, the business logic, and the system itself. Otherwise the estimation will be horrendous.

*Use Cases.* It is noticed that all developers agree that use cases can help them better understand what needs to be built and that they have the most success when they have use cases. One developer mentioned that, “In our Sprint planning session we did a fair amount of documentation in terms of creating use cases. When we had use cases we had the best success.” He continued to explain what his team did. He stated, “We first

received a list of items that we needed to build and then we wrote out specifics, maybe a couple of paragraphs for each item." Another developer added, "I think it really helps us figure out the system that we are going to build."

However, there are several problems in creating use cases. A developer who was not familiar with the system had a hard time creating use cases. He stated, "I felt I was a little unprepared to write use cases for a web site that I was not really familiar with." Another problem is that clients usually do not know the systems that they want to have and what use cases are. One developer stated, "They don't know really what use cases are. So, basically you need to come up with something you think makes sense and write it down and then go back to review it. The idea is that it's really difficult to come up with specifics for it."

Though clients do not know about use cases, it seems that they are comfortable with revising them. A project manager stated, "It's a lot easier for them to revise something we've written." Based on this description, it would be better if clients knew what they really want to have in their system and understand use cases. If this were the situation, developers could communicate better with clients and would create better use cases, which then leads to success of the project.

*Environmental Factor*

The environmental factors including customer involvement, working environment, common tools and problems between teams, government project and Scrum method, and social loafing were identified and are discussed in the following section.

*Customer Involvement.* Customer involvement in the software development process is very critical to the success of the project. The agile methods state that the customer should be part of the development process from analysis and design to implementation and maintenance. However, at XYZ, developers have difficulty working with customers on the projects. A project manager commented that, "Customers are not involved in the decision making process until it is all done." He added, "We don't get as much customer involvement as we want. Our customers are busy and they have other things to do than to talk to programmers all day." One developer complained that, "We request our customers to talk to us every day and at a minimum once a week, but they are not very involved. We end up with talking with them maybe twice per Sprint." Another developer stated,

> "Our customers did not give us specification documents. We basically had an hour-long meeting to make a specification. So it was vague when we started it. It was up to us to make specifics and estimations. I think the biggest roadblock in our development process was in the customer involvement. Though we did not have enough customer involvement, our customers accepted most parta of the system that we created and asked us for minor changes. But I think it would be much better if we get together more often with our customers."

It appears that, most of the time, customers do not know what they really want in their future system and it becomes a roadblock for customers to get involved in the project development process. Said one developer, "Customers think they have a clear idea but they do not. For example, the customer wants to track people's credit. To them, that's clear and precise. But to us, we need to know who the people are, what the credits are, when they expire, how long we track them, what rewards are earned for many credits." Due to unclear customer requirements, developers have a hard time figuring out what exactly the customer wants to include in their system. One Scrum Master

mentioned, “We need to get out a lot of information from unclear statements, which takes more time, which causes us to get involved less because it takes too much time. But we don’t have any other way to do it because we don’t have information.”

*Working Environment.* Most agile methods, including Scrum, recommend removing the cubicles and setting up collocated team space because cubicles promote isolation, and the Scrum process relies heavily on high-bandwidth, face-to-face communication, and networking. Open space is considered better than the cubicles and private offices in the Scrum process. Many developers like the idea of an open-space working environment. One developer mentioned, “I feel like I am little closer to other developers in open space. It’s really nice to be able to look across the room and talk to somebody else in the team and ask questions quickly. I don’t feel like I am shouting over the cubicle wall to get to them.” Another developer stated that, “Open space is good because everyone is easily accessible. I like it because I think it fosters communication. It’s very easy to say hey, I need some help, information, or come, look at this. Everyone is just kind of open, and it seems to work very well.”

Though some developers enjoyed the open-space-working environment, other developers did not like the open space, and they mentioned downsides and some problems. One developer stated that, “the open areas are very nice to communication but it does hurt when you try to concentrate because there are a lot of distractions. For example, when co-workers are having a conversation with somebody or having a phone conversation, it’s very distracting.” Another developer groused that, “I am less productive because a lot of noises are going all around. Without having cubicle walls or private

offices, the distractions are pretty high which is hard to work with." A team lead stated, "You know the best working environment is an office. In your private office, you can do things your way, and focus on things without being distracted by other noises."

To cancel out the noises, most developers use headphones. The director of operations noted that, "Everybody has headphones and they can just put those on and listen to something. That pretty much drowns everything else out." However, several developers complained that, "We developers, are usually working while listening to music. We all have nice headphones. Everything is going under that. But if I need to focus on something, that's really difficult just because I have headphones on."

*Common Technologies and Problems between Teams.* One observation is that one team's members could spend a lot of hours finding the right tools or technologies suitable for their project without knowing that the team next door is already using tools or technologies that could be used for their project. There are many similar technologies can be utilized between teams. One developer mentioned, "A lot of things are similar between teams who are working on different projects, there are even some versions of technologies that we use." He continues by saying that "One is driven by the other so I think it makes sense that we can solve each other's problems using the same tools in many cases." Another developer voiced this thought, "I think that the technologies we use between teams are similar enough that if we do our level best, we are able to cooperate." It is also noticed that some problems that separate teams have are similar. One developer noticed, "I think when problems come up, even though we may not know the languages used in another team's project, a lot of problems are the same."

One of the teams sometimes uses a quite different setup in a different environment, as reported by a project manager, “They are doing things quite differently.” He also added, “I think they will eventually move to some of the technologies we are using. Then we will be able to cooperate more. But now it does make sense that we use a different environment.”

It is noticed that having a person who can play the role of a liaison between teams is important because of the similar technologies used in multiple teams and the similar problems arising with the teams. The person should be able to inform a team whether there are similar technologies that other teams already take advantage of and whether there are similar problems that other teams faced and resolved.

*Government Project and Scrum Method.* Government projects usually require heavy documentation, big planning, and big design up front. Due to the bureaucratic nature of the government, a government project seems not to conform to the principles of the Scrum method. Developers consider that working on a government project is challenging because government itself is not agile. One developer expressed:

> You know here we are really struggling with a government project because they think they are doing agile. But government tends to be very bureaucratic because it tends to be a lot of red-tape and tends to have many layers of accountability inside their organizational structure. It takes a long time for decisions to get made. They are very not agile just by default. And so, trying to get them to function that way can be a challenge.

Another issue in dealing with a government project is that there are many jargons and acronyms used in the descriptions of the government project. Developers have to learn all these unfamiliar terms. They think the government project is complicated. One developer stated, “Of course, it’s affiliated with government and anything with affiliated with

government has a tendency to be complex and then there's a lot of acronyms and there's a lot of vocabulary that I know nothing about." Conflicts between the bureaucratic nature of a government and the principles of Scrum method seem to create a big hurdle in fulfilling the government project. To lower the hurdle, XYZ tries to use the Scrum method with a Unified Process (UP) method, which conforms more to the requirements of a government project. But the hybrid of the Scrum method and the Unified Process method has not been fully developed in the firm.

*Social Loafing.* As explained in ABC's social loafing section, social loafing is an observed fact that people are less productive and less motivated when they work in a group than when they work alone. Also, social loafing is the tendency to take advantage of other's effort when working in groups. It is noticed that most developers feel there has not been much social loafing in their projects. One developer stated, "I don't really see that happening now so much. There are three of us on our team. I think everybody does their fair share of work." Another developer held that social loafing is, "…really not in our team. Everyone is fairly motivated and quick to point out if anyone is not doing his or her share." However, one developer thinks there has been social loafing in his project. He held that, "In our particular project, that's been true to a certain extent." Another developer stated that, "I think that's true sometimes, but I have never been involved in that kind of situation in my project. I haven't got a credit that I think I don't deserve." He also thinks the firm has been very fair in recognizing developers' hard work. He felt that, "They are very good about it. Actually, they are very good at giving credit toward developers who worked hard on a project."

Overall, most developers think social loafing has not been a problem, but some developers expressed an interesting issue. One of them stated:

> I don't think there is so much individual recognition, whereas there is team recognition. So for example, in an XYZ team, when we get the project done, then we recognize the team, rather than 'Oh, John did the most, Amy did the second.' Most stuff like that. As a team, we are all respected as doing an equal amount of work. If you are not part of it, you will get fire.

*Information Systems and Technology Factor*

The information systems and technology factor is comprised of these concepts: communication, bug tracking system, and version control systems. These concepts are discussed in the following section.

*Communication.* The Scrum process recognizes the important role of communications in the software development process and provides an excellent means of communication. All interviewees agree that the Daily Scrum Meetings improve communications between team members. However, each team in the firm is fairly separated and generally there is not much communication between teams. The lack of communication between teams could cause problems, such as duplicated work. This problem can be solved, or at least mitigated, if the firm holds daily Scrum of Scrums meetings so that the Scrum masters from each Scrum team can make sure no work is being duplicated.

Good within-team and between-team communication can be accomplished through the framework of Scrum, but communication with the customer can be problematic. Several developers observed that, "the biggest area of communication issues that we have is with the customer more than anything else, because they tend to not give

us a lot of feedback." Part of the reason that the customer does not provide feedback is that, in most cases, they have other daily jobs to take care of in addition to the work with developers. This is related to the customer involvement issue, which is explained in the customer involvement section.

*Bug Tracking System.* All developers agree on that any web-based bug tracking system can assist them in the task of prioritizing bugs and keeping track of bugs. The company has been using a bug tracking system called "Mantis," which is a free web-based bug tracking system. Most developers think Mantis has been working well for the bug tracking and auditing. One developer mentioned that, "It does a good job as far as bug tracking and auditing." Another developer states that, "I have been pleased with Mantis. We use it very effectively, and it's very customizable."

Mantis also has a search function so that developers can do a key word search. One developer observed that, "Mantis has a good search engine. You can use any subsidiary field, which is a track list, and it can categorize very well." However, some developers think they could use Mantis more effectively by integrating features, tasks, and burn-down charts into Mantis. One developer stated, "We only put bugs into Mantis instead of features and tasks. I want to incorporate burn-down charts into Mantis." A couple of developers also think there is a bug in the software. One developer noted, "There are couples of weird things it does occasionally, but it's nothing catastrophic." The other developer uttered, "Mantis is supposed to send an email whenever a new bug is entered in and sometime it sends false notification. It seems to be a bug in the software."

One interesting thing found in the interviews is that developers want their clients to report bugs into Mantis. A project manager affirmed that, “It would be nice if our clients could get into Mantis and report bugs there.” In fact, a project manager stated that, “Basically, two teams had their clients enter bugs into Mantis. That can be helpful for them.” However, the client in one of the projects does not like using Mantis as a bug report system, so they use an email system instead. This seems to be a problem for developers, because one noted, “they [the client] email the bugs to somebody and often we don’t follow up on them.”

Another issue is that clients’ bug reports are often vague and difficult to understand. One developer mentioned,

> “Usually, their reports are a little vague and hard to understand. They say, ‘I did this and enter the information and then save and then the page went blank.’ Sometimes, you don’t know exactly what page they were in the website and sometimes you don’t know which information didn’t get saved.”

To solve the issue, developers ask clients for more feedback on their reports. One developer stated that, “We can ask our client to put more feedback directly in Mantis and hope our client sees it or sends an email, or we can give them a call if we have any questions on the feedback. It seems that Mantis works well as a bug tracking system for the firm, though it has a little glitch. However, it seems that more effort is needed to integrate features, tasks, and burn down charts into Mantis.

*Version Control System.* The company has been using an open-source revision control system called “Subversion”. All developers commented that Subversion would be a great tool in any environment where a version control system is needed. One developer stated, “We use Subversion and it works very well for us.” Another developer added,

“I’ve been really impressed with Subversion.” Another developer claims that Subversion, “. . . works very well for small teams and for large teams as well.” A project manager who used another revision control system thinks Subversion provides various functions and is superior in many ways. He claimed that, “It has great utility, great tagging and branching. I can only think of pros for it. It has been great for us and you know especially compared to SourceSafe. It’s superior in every way.” Another developer claimed that Subversion makes revision control easy, especially when creating and utilizing a branch. He stated, “I came from using the Concurrent Versions System. Subversion just makes a lot of things easier. We are using Subversion and it has an advantage when you want to create a branch with some project. That‘s the biggest advantage of using it.”

In addition to the advantages mentioned above, Subversion periodically checks if there are any changes made. If there are any changes made in the repository, it rebuilds the system. One developer stated, “We use Subversion. It just polls every five minutes I think and whenever there’s change, it checks out the code and rebuilds it, and make sure everything is still working.”

Some developers think they can use it better by creating more branches. One developer commented, “We need to use it better and do more branching.” Another developer talked about a recent mistake, “Some code got checked in recently into the main branch and got deployed. It caused a problem because it was in a half-ready state. It should have been checked into a branch.” Other than the problem caused by human error, it seems that Subversion is a well-chosen revision control system for the firm, and the firm just needs to expand its use.

# CHAPTER V

# DISCUSSION AND MANAGEMENT GUIDELINES

This chapter discusses the discoveries from an in-depth agile software development study of two firms utilizing the Scrum software methodology. The discussion covers the following topics: critical issues and challenges found by adopting the Scrum method in both of the study firms and management guidelines for avoiding and overcoming obstacles when adopting the Scrum method.

## *Issues and Challenges of Scrum*

This section discusses issues and challenges identified in two firms by comparing concepts which emerged from the data. The issues and challenges discussed suggest lessons that Scrum practitioners can learn and provide a basis for management guidelines.

### *Human Resource Management*

Human resource management comprises in-house people related issues. Table 28 presents issues identified from the two firms. These issues reveal the challenges of the Scrum method.

*Team Management.* When an organization decides to employ the Scrum method, the organization needs to reorganize their existing development teams into Scrum teams. The ABC results indicate that when Scrum teams are created, they must be composed based on the knowledge and skills necessary for the projects in order to reduce the time to learn business logic, development tools, and programming languages. In addition, each Scrum team needs a team leader who can see the big picture and guide the team in the

right direction, even though self-managing team are one of the unique aspects of Scrum. The Scrum master might be a good candidate for this job and selected from a group of technical people, rather than a group of non-technical people like at ABC. Some developers like the idea that the team decides how to do things based on the consensus of the team and that the team has more control over how to complete the development project. When organizing Scrum teams, the organization needs to balance management between a pure self-managing team and a heavy leader-guided team. Another lesson from ABC is that small sized teams are more flexible and adaptable in defining and applying variants of Scrum. Interestingly, team management was not an identified issue at XYZ.

Table 28

*Human Resource Management Issues*

| Category | ABC Firm Issues | XYZ Firm Issues |
|---|---|---|
| Human Resource Management | • Team Management<br>• Collaboration<br>• Training<br>• Lack of Accountability<br>• Trust and Confidence | • Multiple responsibilities<br>• Collaboration<br>• Training |

*Collaboration.* The research identified several collaboration problems at ABC when the firm kept two geographically separated development teams. These problems could also apply to co-located teams. The first collaboration problem occurred between developers at one site and QA personnel at the other site; developers were checking code into CVS without telling QA personnel the potential areas of code that might be affected by the changes made. To mitigate this problem, developers should take some responsibility in testing code, or collaborate with QA personnel by showing the potential areas of code that might be broken by their changes. The cause of the second

collaboration problem was a divided team with members in two different development locations. When the team hired new members at one site, it was difficult for team members at the other site to work with them. This incident indicates that when new members are hired, assign them to a co-located Scrum team instead of a remote-site Scrum team. The third collaboration problem arose between Scrum teams due to the lack of a designated person responsible for checking the consistency of products across Scrum teams. The solution to this problem consists of appointing a Scrum master in each team and having them discuss any common issues across Scrum teams. The fourth collaboration problem concerned dividing and assigning tasks between two sites in the Sprint planning meeting, as well as how to track bugs reported by QA personnel and customers. The mitigation of this problem is to use commercial tools designed to address these issues, such as VersionOne (which offers good project management mechanisms) or JIRA (which provides excellent bug tracking mechanisms).

At XYZ, no major collaboration problems were noticed. On one occasion at XYZ, a situation developed between developers and a particular project manager. Due to insufficient human resources, one person had multiple responsibilities as product manager, product owner, and accounting manager. The first problem was that developers did not try to talk with the individual about issues that came up outside of the Scrum meeting because they knew he was too busy and overloaded with too many other tasks. The second problem was that developers had to wait to get any questions answered. The third problem was the developers had the notion that the overloaded manager was a bottleneck for the rest of team. A fourth problem was that developers had communication issues with him because he was rarely available to people. This situation might be

beneficial to XYZ for a short time. In the long run, however, the firm will lose a lot more than it gains with a manager trying to fulfill multiple conflicting responsibilities.

*Training.* One of the biggest problems ABC faced was in new employee training. Due to the complexity embedded in the system, new employees need to spend a lot of time becoming familiar with the system. ABC deemed training new employees separately in each Scrum team a big waste of time. A more efficient method is to appoint a mentor to train all new employees in the basic practices of the firm, with each team giving task specific training. The problem with new employee training gets worse when the employee who has expertise in an area is at a different site than the new employee. One observation noted that the use of multimedia may help the firm mitigate this problem, but is not a complete solution. The indication is that co-located Scrum team members should do employee training.

XYZ also had a training problem. However, the problem there is with the Scrum development method itself, rather than the existing systems. Some developers did not see the big picture and the benefits of Scrum. One developer had a misconception that he thought he should complete a full project within a single Sprint, instead of multiple Sprints. XYZ's case indicates that it is important to have all new developers go through thorough, step-by-step, formal training and occasional follow-up training after that. Brown bag lunch training or the "lunch-and-learn"-type training offered by the firm to all employees is an excellent way to make up for the lack of formal training and to get feedback from everyone.

*Lack of Accountability.* ABC revealed that tasks in the Sprint backlog often did not get completed as estimated, and were consistently carried over to the next Sprint.

Then, nobody took responsibility for the delayed tasks. The initial intention of a self-managed team is to allow developers to run the team and have ownership of the projects; this seems to have been a big hole at ABC. If not operating correctly, self-managed teams promote a lack of supervision, which can lead to a lack of accountability and commensurate project delays. It appears that it would be helpful for the firm to confer the necessary authority to either project managers or Scrum masters to supervise developers. In that way, project managers or Scrum masters may influence developers to work faster and harder, while allowing developers a certain degree of self-management.

It is also a downside of distributed Scrum that developers do not take ownership of task completion, due to the lack of relevant control at both sites. Batra, Sin, and Tseng (2006) suggested setting up a coordinator in one site and ambassador in the other site to ensure control. A lack of accountability issue was not found at XYZ.

*Trust and Confidenc.:* Trust and confidence arose as an issue between a Scrum master and developers when the Scrum master was unable to get developers items or information they needed and was unable to remove developer's impediments. The same issue arose between developers when developers worked together on the same project and they did not see any progress on a module assigned to another developer or group.

The trust issue got worse in the case of two development sites, causing confidence levels with other developers to degrade. The fifth principle of the agile manifesto (http://agilemanifesto.org/principles.html) says, "Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done." This implies developers should work in an environment that suits them and

should have the support that they need, and at the same time they should give other team members trust to attain high confidence levels.

It takes a lot of time and effort to regain trust and confidence among developers. It is critical to prevent trust and confidence problems before they happen. To resolve a trust and confidence problem, the organization should foster collaboration and have project managers keep monitoring the situation. Also, when there are team members at different sites, it would be helpful if each member of the team at one site would take the time to get to know members at the other site. The trust and confidence was not an identified issue at XYZ.

*Structured Development Process*

The structured development process consists of systematic process related issues. Table 29 shows the structured development process issues identified in the research. They are discussed below.

*Scrum Framework.* Most developers at ABC were in favor of the Scrum framework. They thought the Scrum model promoted communication and team work, and helped them keep track of task assignments and monitor task progress. They also thought that working in the Scrum team provided motivation, excitement, and interest. Some developers, however, did not like the frequent daily Scrum meeting. They felt that having the daily Scrum meeting was too much time for not enough value, and that the various Scrum meetings took too much developer's time away from programming, even though the Scrum meetings helped team members refine the goals for each Sprint and improved the quality of products. Some developers and QA personnel also thought monthly Sprint planning meetings took too much time and they want to streamline the

planning session. As ABC indicates, any organization should have streamlined Scrum meetings and monitor whether or not various Scrum planning sessions take too much time for not enough value.

Table 29

*Structured Development Process Issues*

| Category | ABC Firm Issues | XYZ Firm Issues |
|---|---|---|
| Structured Development Process | • Scrum Framework<br>• Unit and Integration Testing<br>• Coding Standard<br>• Documentation<br>• Formal Code Review | • Scrum Framework<br>• Unit and Integration Testing<br>• Coding Standard<br>• Documentation<br>• Formal Code Review<br>• Project Estimation and Planning Poker<br>• Use Cases |

The Scrum method works quite well for developers at XYZ and is a big improvement over the waterfall method. Developers thought the Scrum method helped team members get involved in projects, be aware of everyone's progress, contain scope creep, and prevent projects from going too far off course. Like ABC, some developers did not like inefficient Sprint planning and review meetings. They felt that keeping daily Scrum meetings to 15 minutes was difficult because people gab a little too long, there is an excessive amount of material that needed discussed, and taking care of 2 or 3 projects at once.

Another problem was setting up the meeting time. It was difficult to get all developers together at one time without interrupting their work because of the flexible work schedule. The research noted that some project managers actually managed the

Scrum team rather than let the team self-manage, and that developers did not spend enough time generating a detailed Sprint backlog.

*Unit and Integration Testing.* When ABC first adopted the Scrum method, the firm placed a QA person on each Scrum team. This caused a problem because the QA person is always behind the Sprint schedule. This problem was resolved by creating a QA-only Scrum team that covered all code generated by other Scrum teams. This solution created another problem that when developers made a change and passed the code to the QA team, the QA personnel sometimes did not know the other areas that affected by the changes. To resolve this problem, the firm asked developers to do some portion of the testing themselves that the QA people usually cover. This may not be a good move economically for ABC firm because developers usually are paid more than QA personnel; this might be a sensitive issue that any organization needs to resolve wisely.

XYZ utilized a tool called “N-Unit” for unit testing, and each developer tested his/her own code. Interestingly, the firm also invited their clients to the test site and had them track, test, and enter bugs that they found. However, having developers test their own code had two issues. First, developers usually assume that their code always worked. Second, developers could not as thoroughly test their code as third-party testers.

Another issue was related to large legacy code not designed for unit or integration testing. Developers had a difficult time testing the legacy code. The legacy code and the code working behind the curtain were a big challenge to the firm. The firm also needed to hire more people who had a wide range of testing skills in software. Insufficient client budget also made it difficult to test everything covering one hundred-percent of the code.

It seems that the firm utilized the unit test well, but it did not cover all possible combinations of issues due the short client budget and lack of wide range of skilled QA personnel. In addition, the firm needed to rewrite the legacy code or find out an efficient way to make the legacy code unit-testable.

*Coding Standard.* ABC utilized coding standards. They have very specific coding standards in many areas in order to have easily maintainable and expandable code. Most developers agree that having a formal coding standard enables them to understand other developer's code, but some developers worry about putting too many coding standards on developer's shoulders. Some developers actually think that heavy coercion to the standard may hamper their performance because they have to look at coding documents back and forth to see if their code conforms to the standard.

XYZ did not have a formal coding standard but had a verbal coding standard; developers felt that they were close enough to comment when a person who did not follow the norm. Developers had their own coding style, which was influenced by several commercial software packages, such as Microsoft Visual Studio and Borland. Most developers and project managers thought forcing a coding standard might hamper developer's creativity and hinder performance because they had to relearn how to code in many places.

*Documentation.* After ABC started the Scrum method, many detail documents, such as class diagrams, sequence diagrams, activity diagrams, communication diagrams, and use cases were significantly reduced, or disappeared. The lack of detailed design, as indicated at ABC, caused many problems in complex projects. One main area affected

considerably was testing, because QA personnel depend heavily on documentation to find problems.

Another problem with a lack of detailed documentation was the tendency to write code without taking time to think about what effects the code may have on other parts of the application. These resulted in an increased number of bugs, which then required a lot of developer working hours to fix. This was a major issue and caused major code re-writes. It's obvious from ABC that if any organizations deal with complex and large projects, they need to tailor the Scrum philosophy on reducing the amount of documentation.

XYZ also reduced the amount of documentations significantly. Developers tried to place more comments and explanations for any tricky logic in the code, along with explanations for any changes that they made, to compensate for the lack of documentation. However, it turned out that many developers had a hard time completing tasks without any documentation, especially developers who needed to work on parts of the system they had never worked on before and new developers who did not have much experience with XYZ's projects. Further, those developers asked a lot of questions, which took much time away from developers who did understand the project. As XYZ indicates, no documents at all are a very dangerous idea that leads to many problems, including causing the agile method to be as slow as anything else.

In the agile methods, the code itself is regarded as all the documentation that developers need. However, it is apparent that zero documents are not always the right way for large-scale and complex projects, especially, in a distributed Scrum environment. The amount of documentation should be decided based on the context of the development

environment, though Parnas (2006) suggests a wordy document and Simon (2006) suggests no more than a two page long document**.**

An additional problem was that only one main developer had extensive knowledge about the firm's systems, rather than every developer on the Scrum team having shared skills and knowledge of the systems. If that main person leaves the firm for any reason, it would be a big problem because it may take several months to recover the knowledge lost. Keeping all team members equal by sharing skills and knowledge on the systems is not easy, and not feasible in reality.

*Formal Code Review.* ABC has utilized a web-based formal code review, and developers think the formal code review is a vital and critical process in creating high-quality software applications. Any issues and challenges were not identified at ABC.

Developers at XYZ did not have a formal code review, but they had an occasional informal code review. Not having a formal code review invoked some issues. First, developers did not pay extra attention to their code, because they believed no one would look at it again. If they believed that at some point somebody would go back and look at their code, they would have more accountability. Second, developers lost opportunities to improve the quality of their code and enhance their coding skills through feedback from other developers. Third, there was a high chance that developers wasted time and money by trying to re-invent the wheel from scratch because there are many code examples already written, tested, and proved efficient by the Dot Net framework or other commercial builders. Most developers knew the benefits of having a formal code review, but they just never had the time to do it. XYZ needs to set a time aside for a formal code review and select an appropriate tool to facilitate the code review.

*Project Estimation and Planning Poker.* This issue came up only at XYZ, though having an accurate project estimate was an important part of projects for both firms. Developers at XYZ had a hard time estimating the duration of a project, and the level of hardness increased when developers needed to deal with legacy code or when they did not have the experience required to finish a project. However, it seems they mitigated this issue by introducing a new project estimation method called "Planning Poker". A lot of comments from developers revealed that the Planning Poker method provided developers with the opportunity to throw out their honest opinion without being biased or coerced by other developers. Also, Planning Poker helped developers have a chance to discuss estimation gaps between developers, and guided them to reaching better estimates. One noticeable benefit was when developers were able to break big tasks into the smallest measurable segments, they were easily able to make good estimates. The researcher noticed that every developer thought Planning Poker was very useful, effective, and produced reasonable estimates. However, though Planning Poker could help estimation, the bottom line was developers needed to be familiar with the technologies that they were going to use, the business logic, and the system itself. Otherwise, estimation will still be one of the most difficult parts of a project.

*Use Cases.* This issue was identified only at XYZ. Developers at XYZ knew that they could understand the system better with use cases and that they had the best success when they had use cases. Though the firm has reduced the amount of specification documents a lot since the firm adopted Scrum, one Scrum team created a fair amount of use case documentation based on a list of items that team members needed to build. Three issues were identified related to creating use cases. First, some developers were not

well prepared to write use cases because they were unfamiliar with a system. Second, clients did not have a clear and precise idea what they really wanted to have in their system. Third, clients did not know what use cases are or how to use them.

The first issue is an in-house issue and the other two are client-related issues. The in-house issue can be resolved through a well-organized employee training program. The client-related issues were resolved by having developers come up with some specifications and having clients review it. It would be better if clients know what they really want to have in their system and understand use cases. If clients have a notion of use cases and have a clear idea of their system, developers can communicate with clients better and create better use cases, which can lead to successful projects.

*Environment*

Table 30 shows issues belong to the environmental factor. The issues and challenges related to the environmental factor are discussed below.

Table 30

*Environment Issues*

| Category | ABC Firm Issues | XYZ Firm Issues |
|---|---|---|
| Environment | • Customer Involvement<br>• Working Environment<br>• Interdependency among Modules<br>• Social Loafing | • Customer Involvement<br>• Working Environment<br>• Common Tools and Problems between Teams<br>• Government Projects and the Scrum Method<br>• Social Loafing |

*Customer Involvement.* Due to the large number of customers scattered across the United States, ABC needed to come up with a different solution to incorporate customer feedback. One way that the firm employed was to send out product line managers to the

customer and have them collect project requirements. The project line managers also utilized WebEx to show features of the products and some charts and graphs to reduce the number of onsite visits. Another way the firm employed to get customer feedback was to host a user conference once a year. At the conference, the firm demonstrated new policies and directions of product development. The customers then voted for or against the policies and new development direction.

One issue associated with the customer involvement at ABC was that QA people had a difficult time providing the customers with quick bug fixes. This resulted from the Scrum method principles, which required the QA people to focus more on the code generated during the Sprint process than on responding to customer problems.

Though each project at XYZ was only for one customer, developers at XYZ had difficulties getting customers involved in the decision making process. The customers did not willingly participate in the process because they were busy and had other things to do. Poor customer involvement in projects caused problems for the firm because developers needed to create specifics without conversing with clients. Often times it took a lot of hours to figure out what exactly customers really wanted to include in their system, because they did not know what they want in their future system. This was a big roadblock for developers as they went through the development process.

Though XYZ did not have enough customer involvement, most final products were accepted by their customers with minor changes requests. XYZ's case indicates that if developers get together more often with their customers, organizations can deliver a software product to customers sooner with better features and functions. Organizations

may also reduce maintenance fees by delivering a more correct product that customers want.

*Working Environment.* There were mixed feelings among developers about the open-space working environment. Some developers liked a cubicle setting because they thought it increased the number of communications between team members and fostered collaboration and the teamwork. However, most developers did not like the open-space working environment because they could not concentrate on their work while their coworkers talked to one another. It was apparent that most developers liked having their own office rather than a cubicle in order to be productive.

Many developers at XYZ liked the open-space working environment because it provided easy access to other developers and it fostered communication. Though some developers enjoyed the open-space working environment, other developers did not like it and thought it brought some downsides and problems. First, developers were easily distracted when their co-worker's talked to other co-workers or when they had a phone conversation with someone. Second, developers were less productive when they could not concentrate because of a lot of background noise. To cancel out the noise, developers utilized headphones, which they put on to drown everything else out. Though this helped most developers, some could not focus on their works just because they have the headphones on.

*Interdependency among Modules.* At ABC, as the size and complexity of the project grew, the dependencies and interconnections among tasks in the application increased. However, developers were not able to fully consider all the dependencies and interconnections among modules because of their narrow-focused planning and design in

each Sprint planning meeting. The developers also had a tendency to do things in a quick and dirty way without thinking whether the code would be flexible enough for future needs. As ABC indicates, any organizations should support and encourage developers to spend more time on considering the dependencies and interconnections among modules. The issue of interdependency among modules was not identified at XYZ because XYZ's projects were relatively small and less complex compared to ABC's projects.

*Common Tools and Problems between Teams.* Though XYZ did not show any signs of interdependencies among modules due to the firm's small size of projects, the firm did have issues with common tools and problems between teams. It appeared that one Scrum team's members could spend many hours finding the right tools or technologies suitable for their project without knowing that other Scrum teams already employed similar tools or technologies. This is a big waste of precious developer's time if two Scrum teams can utilize the same or similar tools or technologies.

The research also noted that each Scrum team had similar problems, which might be resolved using similar solutions. Teams spent time resolving similar problems, each in their own way, which is another waste doing duplicate work if the same solution can be applied to both problems. The firm should appoint a person to inform teams if there are similar technologies that other teams already took advantage of and whether there are similar problems that other teams faced and resolved successfully.

*Government Project and the Scrum Method.* This issue was identified only at XYZ because the firm has been dealing with many government projects. A government project usually requires heavy documentation, big planning, and big design up front. This does not conform to the philosophy of the Scrum method. It is a big challenge to

complete government projects with the Scrum method because the government itself is not agile, and the nature of government is bureaucratic. An additional issue is that developers have to learn the jargon and acronyms used in the descriptions of government projects, adding unproductive time to complete a government project. To overcome these hurdles, XYZ wants to combine the Scrum method with a Unified Process (UP) method in order to conform more to the requirements of government projects. This hybrid of Scrum and the Unified Process has not been developed in the firm. The author of this paper developed a possible hybrid method as a part of this research, which is explained in the Future Study section.

*Social Loafing.* The social loafing issue was identified at ABC because the firm did not have ways to evaluate individual performance. In addition, social facilitation, group motivational gain, and evaluation apprehension were also found. Social facilitation and group motivational gain contributed to reduced development time and to lower bug rates. When the firm invited all developers, QA personnel, and people in other departments to the Sprint review meeting, developer evaluation apprehension was observed.

Social loafing was generally not an issue at XYZ, though it was present to a certain extent in some projects. Though XYZ has been very fair in recognizing developers' hard work, some developers did not think there was much individual recognition, rather they thought there was more team recognition.

If the firm does not provide a way to accurately measure an individual's performance, and the performance is measured only by the unit or team, a social loafing issue within the team might be raised. It is import to eliminate social loafing by making

each individual's contribution verifiable (Balijepally, 2005) and to offset its possible negative impact on development time and cost. It might be a good idea to invite all developers, QA personnel, and staff in other departments to the Sprint review meeting, where developers present what they implemented. In this way, developers might gain an evaluation appreciation of how others view their work.

*Information Systems and Technology*

Table 31 shows issues associated with information systems and technology. Those issues and challenges are discussed below.

Table 31

*Information Systems and Technology Issues*

| Category | ABC Firm Issues | XYZ Firm Issues |
|---|---|---|
| Information Systems and Technology | • Communication System<br>• Information and Knowledge Sharing System<br>• Bug Tracking System and Management Tool | • Communication<br>• Bug Tracking System<br>• Version Control Systems |

*Communication System*. Ineffective communication is the root of most failures in software products (Parnas, 2006; Simon, 1990). ABC utilized a lot of communication devices to set up good communication channels when they kept two geographically different development sites. Some of the multimedia tools utilized by the firm included video conference systems, phone conference systems, a web-demo system, an instant message system, and an email system. A virtual private network system was also utilized at the beginning, but it was discarded because it was not fast enough. Of the different multimedia tools used, the video conference system was most effective because people could see facial expressions, gestures, and body language as issues were discussed.

However, none of the multimedia systems worked as optimally as face-to-face conversation. Beck et al. (2001) stated that the most efficient and effective method of conveying information to and within a development team is face-to-face conversation. A contributing problem was unstable communication and data lines. When the phone systems were down, the data line went down with it because they shared the same physical line.

Overall, the Scrum method provided an excellent communication mechanism at XYZ. Developers believed that the Scrum method improved communications considerably between team members. Each team in the firm was fairly separated and generally there was not much communication between teams. The lack of communication between teams caused problems as explained in the issues of common tools and problems section. This problem might be easily resolved by holding a daily Scrum of Scrums meeting and having Scrum masters of each Scrum team communicate with each other.

Another communication issue was with customers. More than anything else, this was the biggest communication issue the firm had. As described in the issues of customer involvement section, customers tended not to communicate with developers and not to give a lot of feedback because they usually had other pressing work to do.

*Information and Knowledge Sharing System.* At ABC, newly hired developers created a lot of bugs because they were not knowledgeable about the software that they were working on. Some parts of the application were very sensitive to changes, so a slight modification on these parts had a lot of impact in other areas. It is important to have well-structured information and knowledge-sharing systems between experienced software developers and brand new software developers. In particular, it was very critical

to have a knowledge sharing system if new software developers in one location need some of the expertise of software developers in another location. To resolve this problem, the firm utilized a web-based Wiki program which enables developers to add and edit items that might be critical to other developers. Another issue was assigning newly hired developers to Scrum teams. Though the firm maintained two geographically different development sites, each Scrum team's members should be co-located in only one site. In that way, new developers can have face-to-face communication with other team members.

*Bug Tracking System and Management Tool.* ABC developed a bug tracking system called MOM, which ran on a UNIX platform. Later, the firm adopted a commercial bug tracking system called JIRA because of its additional functionalities, such as filter functions and severity priority code functions. The firm also utilized a commercial web-based management tool called VersionOne to provide useful functionalities, such as simplifying project planning and management, enhancing business and project adaptability, improving project visibility, and increasing project predictability and confidence. The problem associated with these tools was that the firm could not use one product for both bug tracking and management of a Scrum cycle. Developers had a difficult time using all three products at the same time.

XYZ utilized a free, web-based bug tracking system called "Mantis," which worked very well for bug tracking and auditing purposes. However, developers thought they could use Mantis more effectively by integrating features, tasks, and burn-down charts into Mantis. Mantis sometimes also sent out false email notifications when a new bug was entered in the system. Customers associated with two Scrum teams entered bugs

into Mantis, which was very helpful for developers because every bug could be managed by one tool.

Customers associated with other Scrum teams reported their bugs through an email system. This sometimes caused a problem because developers needed to deal with two systems to track bugs, and they did not follow up on customer's emails. Another problem was vague customer bug reports, which developers had a difficult time understanding. This took a lot of developer time because they had to take extra steps deciphering the customer report. Developers usually asked customers to put in a more detailed description, sent an email, or called directly to get more feedback on the report.

*Version Control Systems.* XYZ utilized a version control system called "Subversion," a well-know version control system in the open source community. It has been a great utility for the firm because of its superior tagging and branching capability and a functionality that checks periodically to see if there are any changes made in code and then rebuilds whole system if any changes are detected in the repository. In addition to the automatic rebuild capability, it also makes sure every piece of code is still working properly. Some developers think they can use it better by creating more branches and expanding its use. Other than a human error problem that checked some code into the wrong branch, the firm picked a good revision control system.

### *Management Guidelines*

This section provides management guidelines to help organizations that are already utilizing Scrum or planning to implement Scrum in the future. The guidelines explained here also help organizations avoid stumbling blocks in their Scrum

implementation. The issues and challenges identified and discussed in the previous section provide the basis of the guidelines. The first set of guidelines is for co-located Scrum teams, and the second ones for geographically scattered Scrum teams.

*Guidelines for Co-located Scrum Teams*

1. When a new Scrum team is organized, managers must consider whether each team member's knowledge and skills are pertinent to the project the team members are going to work on.
2. Each Scrum team needs a team leader who can show team members the big picture and guide them in the right direction. The Scrum master might be a good candidate for the team leader. The team leader should not manage the team, but should instead coach the team. The Scrum master should be a technical person able to easily remove technology-related developer impediments.
3. When developers check code into a code repository, they should inform QA personnel, in addition to the direct code that the developers modified or added, about any other code sections that need to be tested as a result of the modifications.
4. The project manager should not be a bottle neck to Scrum teams due to his/her multiple responsibilities in other areas that are not directly related to the team project.
5. Formal step-by-step new employee training should be a requirement, and managers should monitor if the training is efficient.
6. New employees should be given enough time to understand both the existing systems and the Scrum method before they get into a project.

7. Brown bag lunch training or “Lunch-and-learn” type training should be implemented often to refresh developers on Scrum and to exchange information.
8. A self-managing Scrum team still needs a supervisor who has authority to get developers to work faster and harder.
9. Project managers should monitor if there are any trust and confidence issues among developers, and between Scrum masters and developers.
10. The duration and rules of the daily Scrum meeting should be strictly observed; the duration of other Scrum meetings should be dynamically adjusted based on the agenda for efficiency.
11. Project managers should foster collaboration between developers and QA personnel. Developers should be able to do a unit test of other developer’s code and work closely with QA people on integration testing.
12. Organizations should educate developers that every piece of code should be testable and designed for ease of testing.
13. Formal coding standards increase readability and understanding of other developer’s code; too many coding standards hamper developer performance.
14. Lack of documentation is a source of problems, especially for large-scale and complex projects. The philosophy of the Scrum method which reduces documentation significantly should be tailored. Organizations need to determine how much documentation is adequate for their projects.
15. Along with documentation, organizations need to promote each Scrum team member having an equal amount of skill and knowledge relative to the project they are working on.

16. Formal code review is a vital and critical process for quality applications. Organizations should establish an efficient way to do formal code reviews.
17. Planning poker is a very easy, useful, and efficient way to evaluate projects. Developers should break a big project into the smallest possible tasks to get better estimates on those tasks.
18. Use cases are important specifications that elevate a developer's understanding of the project that they are working on; both developers and clients need to be educated in how to write good use cases.
19. Customer involvement is very critical for the success of a project. Organizations should invite customers to participate in the decision making process and find out a good way to include them in the various Scrum meetings.
20. If any organizations have a large number of scattered customers, they should consider the use of an annual or semi-annual user conference to explain their new products, collect user feedback, and have them vote for or against the organization's new product direction.
21. Open-space working environments promote teamwork and communication, but organizations should come up with methods to help developers deal with environmental distractions.
22. For large-scale and complex projects, organizations should encourage and support developers spending sufficient time thinking about dependencies and interconnections between modules.

23. If any projects require heavy documentation, big planning, and/or big design up front, the Scrum method might not work well unless combined with another method, such as a Unified Process.
24. Organizations should provide a fair way to measure individual performance to prevent social loafing being an issue.
25. A product fair, which invites people in other departments to a Sprint review meeting and have developers present their works, is an excellent way to boost developer performance.
26. Bug tracking systems and project management tools should be combined into one piece of software to improve efficiency.
27. If possible, invite customers to a test site and have them test code and enter their bugs into the organization's bug tracking system, rather than letting them email bugs to developers.
28. Good version control systems should be established and utilized to maintain various branches of each product.
29. A person should be selected in each team to work as a liaison between Scrum teams to discuss any common issues and common tools that can be shared among Scrum teams. The Scrum master of each team might be a good candidate, and daily Scrum of Scrums meetings need to be utilized for this purpose.

*Additional Guidelines for Geographically Distributed Scrum Teams*

1. When developers check code into a code repository, they should inform QA personnel at the other site, in addition to the direct code that the developers

modified or added, about any other code sections that need to be tested as a result of the modifications.

2. Scum team members should be co-located at one site, rather than having a team comprised of team members from two different sites.
3. There should be a person to work as the liaison between sites. This person should facilitate collaboration when dividing and assigning tasks between sites. The project manager at each site might be a good candidate.
4. There should be good bug tracking tools and project management tools, accessible and shared by multiple sites.
5. New employee training should be performed through members at the same development site.
6. Project managers at each site should pay extra attention to trust and confidence issues between developers at different sites.
7. Trust and confidence issues are reduced if each Scrum team member at one site takes time to get to know members at the other site.
8. A video conference system between sites works best for communication between sites, but does not work as optimally as face-to-face conversation.
9. Tools like a web-based wiki program should be utilized to share information and knowledge between developers at different sites.

# CHAPTER VI

# A THEORETICAL MODEL, FUTURE STUDY, AND CONCLUSIONS

This chapter suggests a theoretical model for the adoption and utilization of agile methods in the development of mission-critical, small- and large-scale projects and a new hybrid model for further research on the application of traditional and agile methods. This chapter also presents limitations of the present research and lastly some final conclusions.

## *A Theoretical Model*

In the previous chapter, four critical factors for the success of Scrum were described. This section addresses relationships among the four factors and managerial insights to maximize the benefits of software development with Scrum. This section also provides a theoretical model to explain how agile methods can be adopted and utilized to effectively support the development of mission-critical, small- and large-scale projects.

The first factor was human resource management to reflect the importance of the team composition, collaboration, training, accountability, trust and confidence and multiple responsibilities. In terms of human resource management, the organization should consider what would be the optimal allocation of a limited number of developers. It was obvious that it took more development time and cost when new team members did not interact well with other members of the team (Williams & Kessler, 2000) or if team members needed more learning and training to complete their tasks. The organization should foster collaboration between developers and QA people so that QA people can test changed code and other areas of code that might be affected by code changed.

Assigning an appropriate and technical Scrum master is also important for the success of the Scrum method. Sometimes team impediments were not taken care of and often needed to be repeated, slowing or even stopping the progress of the Sprint. Scrum masters should elevate trust and confidence levels among developers and between developers and themselves.

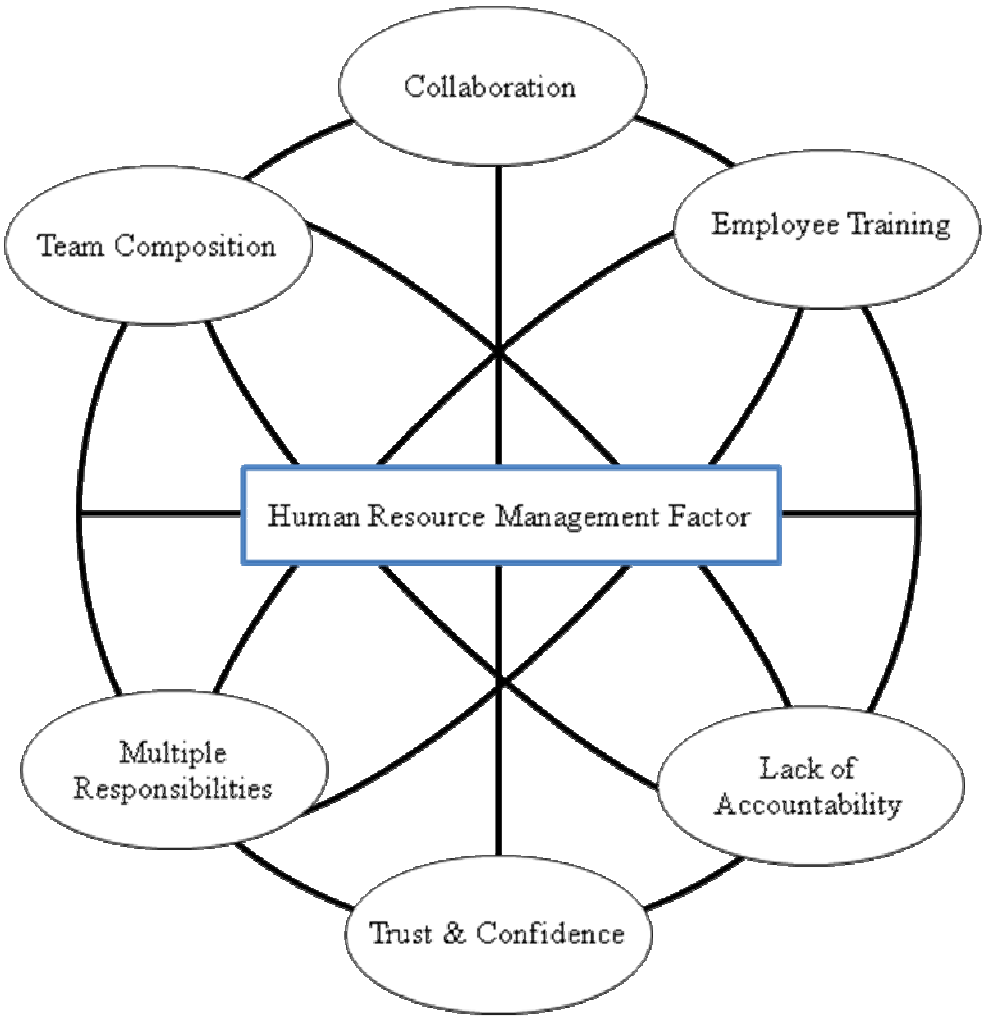



*Figure 10*. The theoretical model of a human resource management factor.

Scrum masters also need to have authority to urge developers to work faster or harder. It may not be a good idea to have developers run the team without having a team

leader (Marrington et al., 2005) to make right decisions in a timely manner when team members do not know which way to take among several alternatives.

New employee training should not be treated lightly. The organization should provide step-by-step formal training with the organization's systems and software development method to new employees. If developers need to do other projects asked by other departments or project managers have multiple responsibilities, they cannot focus on current project tasks and Scrum won't work. In conclusion, human resource management issues can directly affect the performance of Scrum teams. Figure 10 show the theoretical model of the human resource management factor. All of the issues and challenges related to the factor should be reduced or resolved to successfully manage human resources.

The second factor is a structured development process to reflect the importance of the Scrum framework, unit and integration testing, coding standards, documentation, formal code review, project estimation and Planning Poker, and use cases. In terms of a structured development process, all Scrum frameworks, including Scrum ceremonies and Scrum artifacts, should work together smoothly. The organization should also carefully examine the role of a QA person on each Scrum team or on a QA-only Scrum team. QA people can conduct unit and integration testing, and relieve the burden of developers testing their own code. Having a formal code review and coding standards are critical, but organizations should not put too much weight on developer shoulder. The organization should identify how much documentation is appropriate for each project based on the context of the development environment. The amount and utilization of use cases should also be determined on the same context. Project estimation using Panning Poker can be

easily implemented, but the results will be outstanding. Figure 11 displays the theoretical model of a structured development process factor. Each process should be properly established and wisely used.

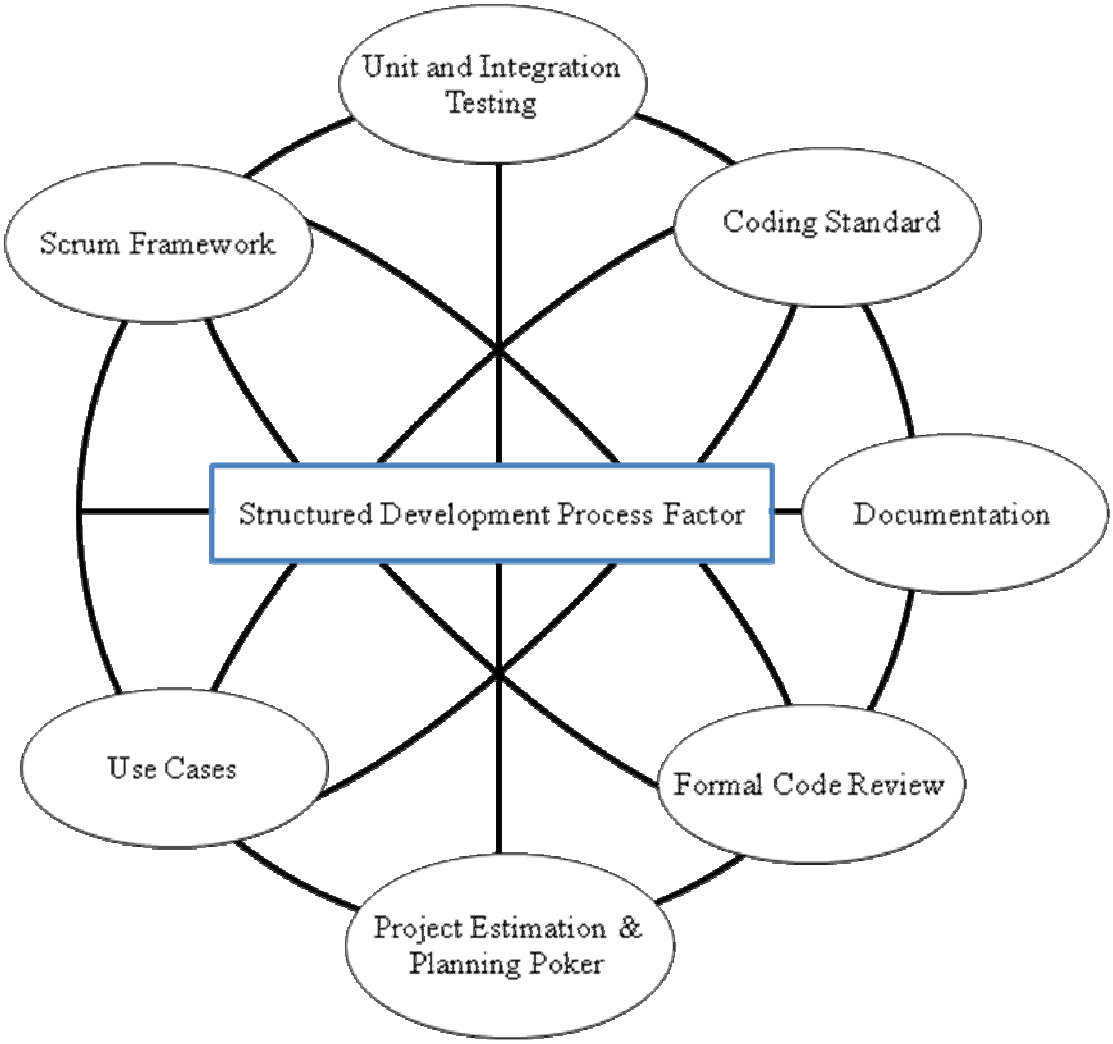



*Figure 11*. The theoretical model of a structured development process factor.

The third factor is the environmental factor, reflecting the importance of customer involvement, working environment, interdependency among modules, social loafing, and common tools and problems. In Scrum, the autonomous nature of Scrum team allows

developers to have more control over how and when the development is completed, depending on the consensus of the team, and to have more ownership of the projects they are working on. Therefore, it is tempting to state that an autonomous Scrum team has better control and efficient product management. However, often the developers are not able to fully consider all the dependencies and interconnections among modules, resulting in inconsistent product outputs across team members within the same Scrum team.

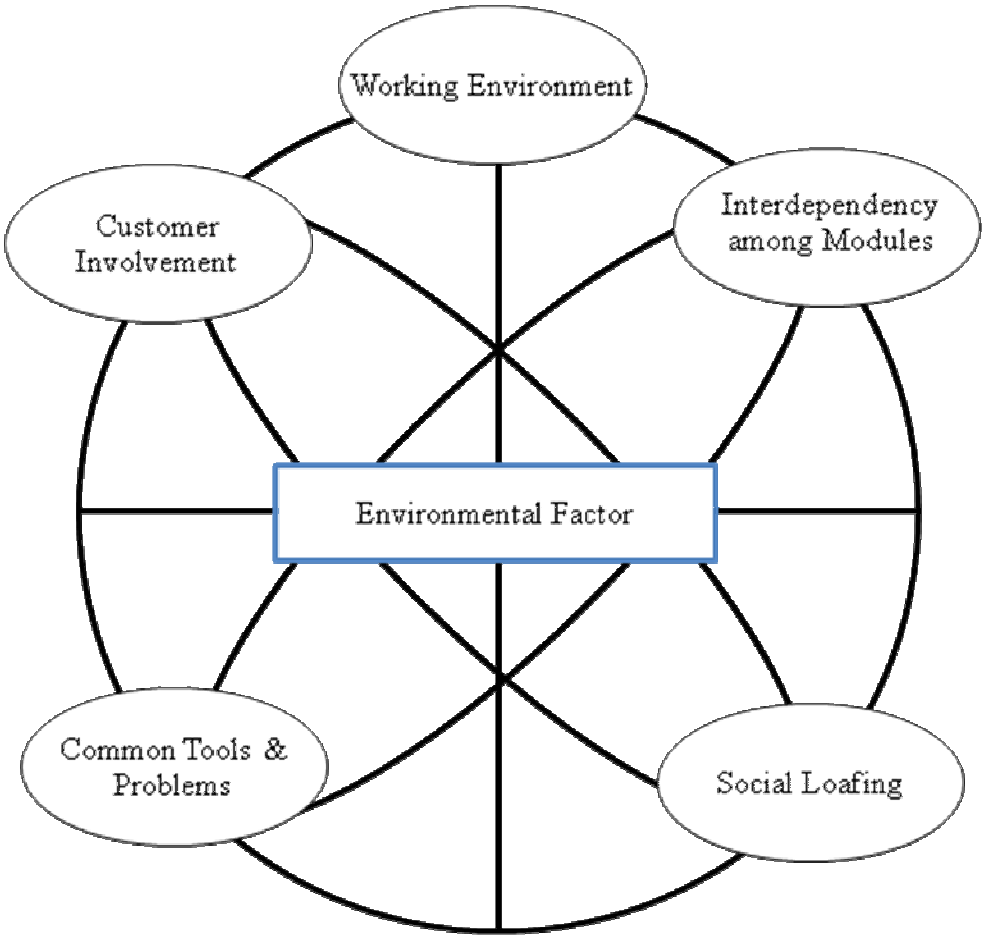



*Figure 12*. The theoretical model of an environmental factor.

Further, due to limited communication between different Scrum teams, it is difficult to keep the overall product output consistent across teams. The lack of

communication results in them solving common problems, each in their own style and way, though the same solution can be applied to common problems across teams. The inconsistency and duplication across teams negatively affects efficient product management.

Regarding the large number of customers scattered in broad areas, a better model should be developed for collecting user requirements and feedback. The research noted that while sharing cubicles with a co-worker improves the opportunity to communicate among team members, it is possible for developers to be constantly distracted by cubicle partners often having conversations with other coworkers. If the organization provides a way to accurately measure the individual's performance, a social loafing issue within a team might be diminished. Figure 12 illustrates the theoretical model of the environmental factor.

The fourth factor is information systems and information technology, to reflect the importance of communication system, information and knowledge sharing system, bug tracking system and management tools, and version control systems. These systems and tools are indispensable for the success of Scrum.

The Scrum method envisions autonomous teams who are given a strong motivation to iteratively and continuously monitor the progress of their projects through interactions and discussions in the daily Scrum meeting and Sprint review meetings. Communication systems, bug tracking systems, and management tools are beneficial in the sense that they help developers visually see and remember what needs to be done on a daily and monthly basis. Figure 13 shows the theoretical model of the information systems and information technology factor.

The final theoretical model, which combines the critical factors, is presented in Figure 14. As explained in the previous sections, all of the issues and challenges in each category should be resolved, or at least mitigated, and the four factors should be balanced and support each other to have success when Scrum is applied to mission critical, small- and large-scale projects.

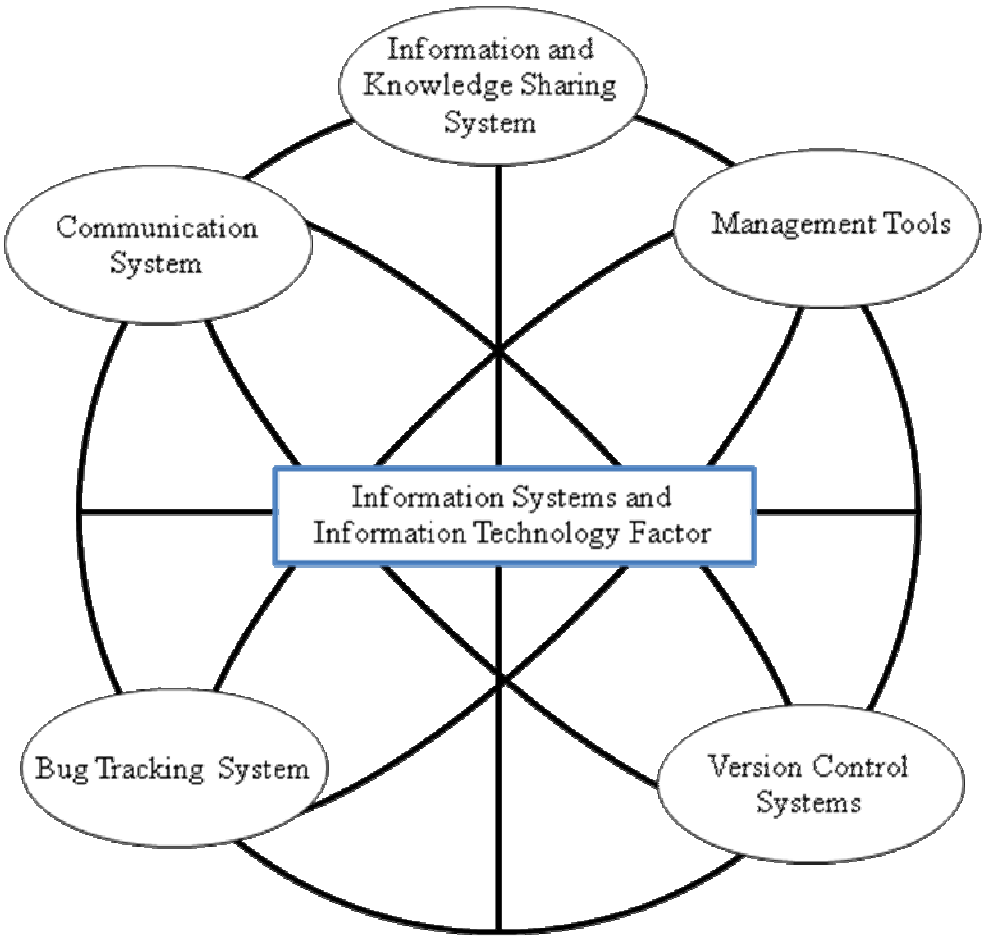



*Figure 13*. The theoretical model of an information system and information technology factor.

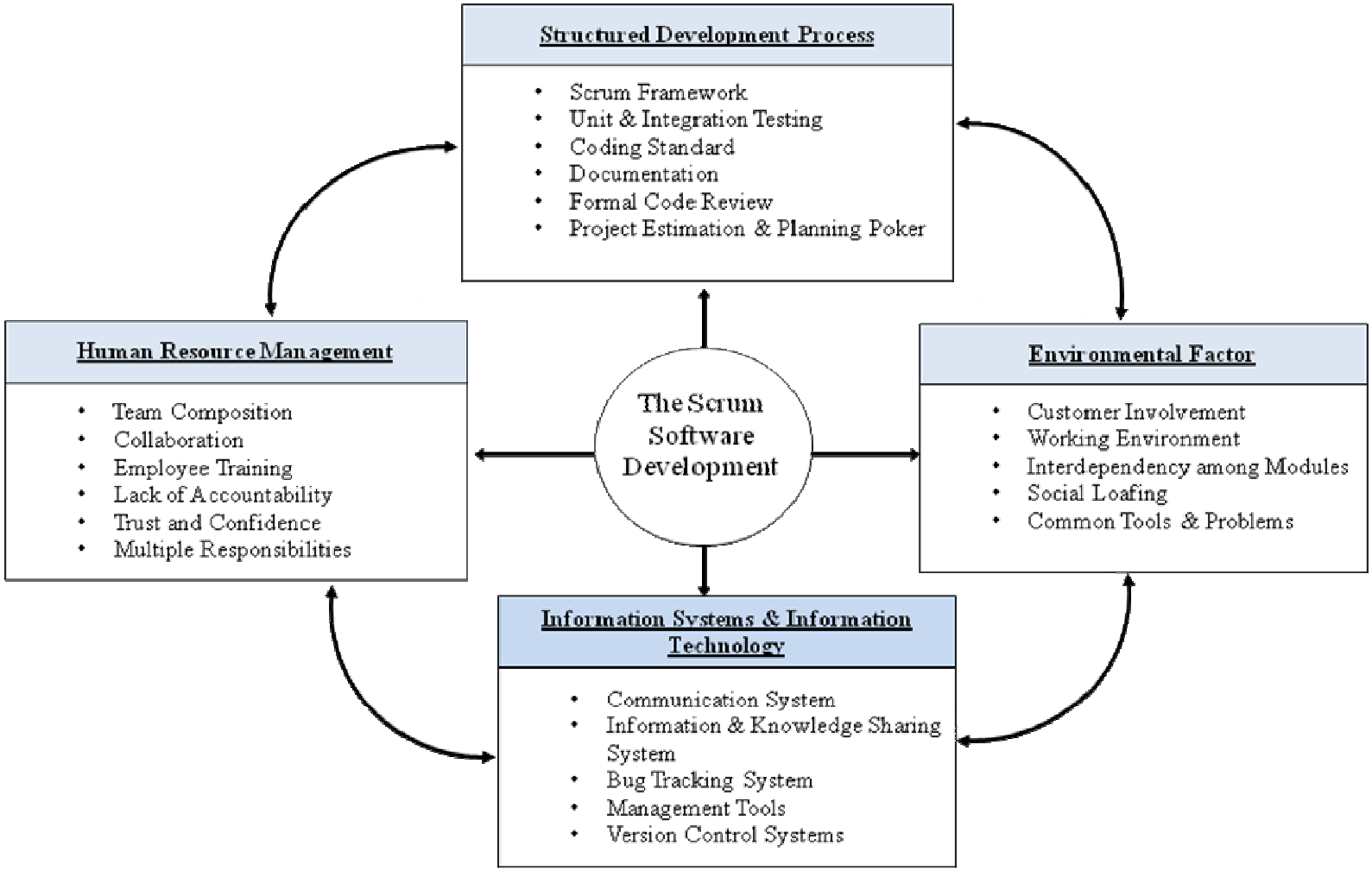



*Figure 14*. Theoretical model for the success of Scrum.

*Future Study: A Hybrid Model*

As described in the literature review section, agile methods have mainly been utilized in relatively small-scale and simple projects and have not been sufficiently tested in large-scale projects, although researchers have reported that large-scale and complex projects also benefited from suitably tailored agile development methods (Bowers et al., 2002; Cao et al., 2004; Lindvall et al., 2004; Lippert et al., 2003). Both agile methods and traditional methods have strengths and weaknesses as shown in the literature review section. It would be very beneficial if we can come up with a new method that accommodates the strengths while suppressing the weaknesses of both methods.

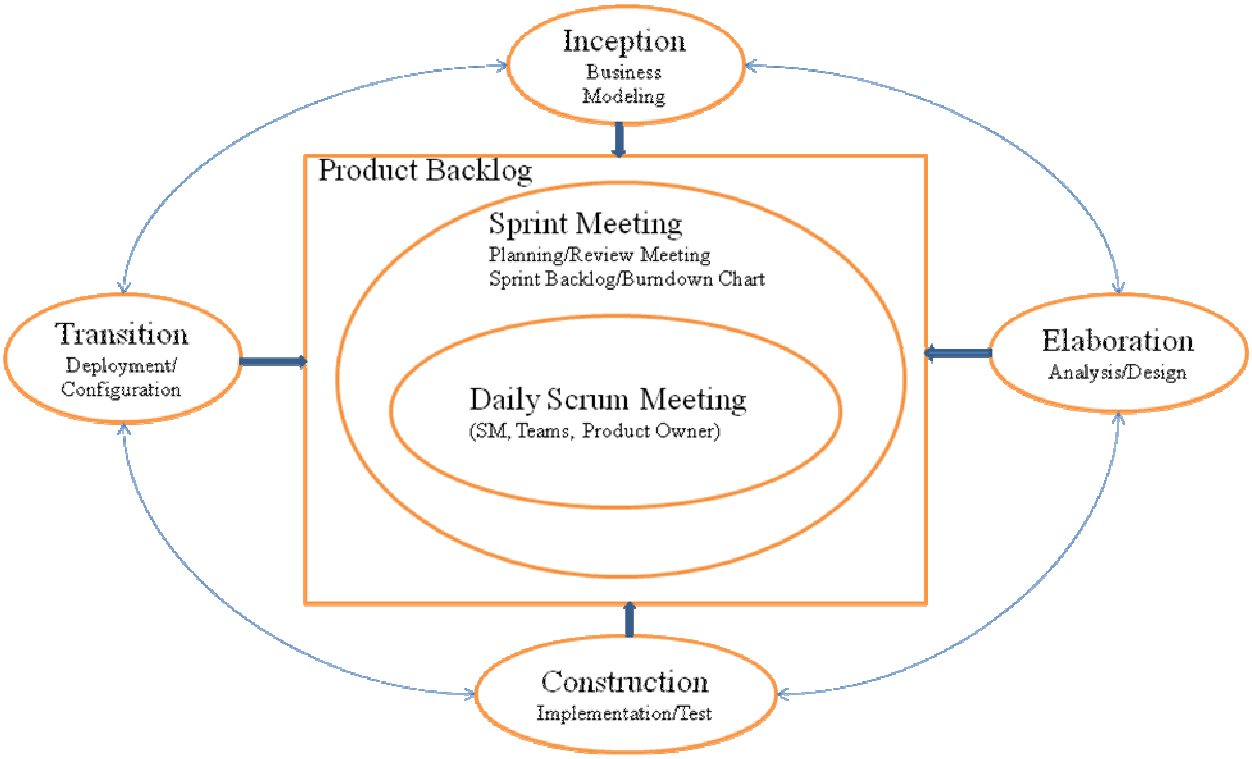



*Figure 15*. A new hybrid model.

One solution for this issue is to create a new hybrid method by combining the Scrum method with the Rational Unified Process (RUP) method. The rationale of the selection of the Scrum method from among other agile methods is: (1) Scrum is a widely used agile method in the software industry, in particular in the United State (Leffingwell, 2007; Williams & Cockburn, 2003), (2) Scrum is powerful and easy to learn (Willson, 2009), and (3) the Scrum method claims to be suitable to any size of projects (Schwaber & Beedle, 2002). Also, RUP is relatively easy to streamline (Ambler, 2005). As a part of the research, the author created a new hybrid model of RUP with Scrum. Figure 15 displays the new hybrid model.

Table 32

*Disciplines of Hybrid Model and RUP*

| Dimensions | | RUP | Hybrid |
|---|---|---|---|
| Phases | | • Inception<br>• Elaboration<br>• Construction<br>• Transition | • Inception<br>• Elaboration<br>• Construction<br>• Transition |
| Disciplines | Main Disciplines | • Business Modeling<br>• Requirements<br>• Analysis & Design<br>• Implement<br>• Testing<br>• Deployment | • Business Modeling<br>• Analysis & Design<br>• Implement<br>• Testing<br>• Deployment |
| | Support Disciplines | • Configuration & Change Management<br>• Project Management<br>• Environment | • Configuration & Change Management |

As shown in Figure 15, the four major phases and disciplines of RUP provide the skeleton of the new method. The nine principles of RUP are reduced into seven

disciplines to streamline the process. All of the seven disciplines can be utilized in each phase, but only the main disciplines are displayed in Figure 15. The business modeling discipline is the main player in the inception phase. The analysis and the design disciplines are mostly utilized in the elaboration phase.

The implementation and testing disciplines focus on the construction phase, whereas, the deployment and configuration disciplines are in the transition phase. Table 32 shows the seven disciplines of the hybrid model along with the original nine RUP disciplines

The ceremonies (Daily Scrum meeting and Sprint meeting) and roles (SM, team, product owner), and artifacts (product backlog, Sprint backlog, and burndown chart) of Scrum can be embedded into the RUP phases without causing any trouble. The daily Scrum meeting, the daily Scrum of Scrums, the Sprint planning meeting, and the Sprint review meeting can be conducted iteratively in each RUP phase. The product owner can create the product log as a part of the business modeling discipline. A Scrum master also can play the usual role defined in the Scrum process. The tasks defined in the product backlog and the Sprint backlog can be accomplished and monitored through the daily Scrum meeting and the Sprint meeting.

Figure 16 illustrates a typical phase of the hybrid model. As shown in the figure, the left-most column contains the product backlog and burndown chart. These Scrum artifacts can be shared by multiple Sprints, which are displayed on the top of each column. Each Sprint starts with the Sprint planning meeting and ends with the Sprint review meeting. The seven RUP disciplines can be monitored through the daily Scrum meeting. Based on the progress of a project, the usage of the seven RUP disciplines will

vary in each Sprint. For example, in the inception phase, the first Sprint may focus more on the business modeling discipline than the other disciplines. However, the second and third Sprint may utilize the analysis/design and the implementation/test disciplines. Figure 16 illustrates the inception phase, which consists of multiple Sprints, but each phase can contain only one or two Sprints according to the size of a project.

As shown in both Figure 15 and Figure 16, we can still provide a straightforward, methodical, and structured process in our hybrid method by keeping the four major phases of RUP. However, the hybrid method will lose some degree of predictability, stability, and high assurance because of the agility of Scrum embedded into RUP.

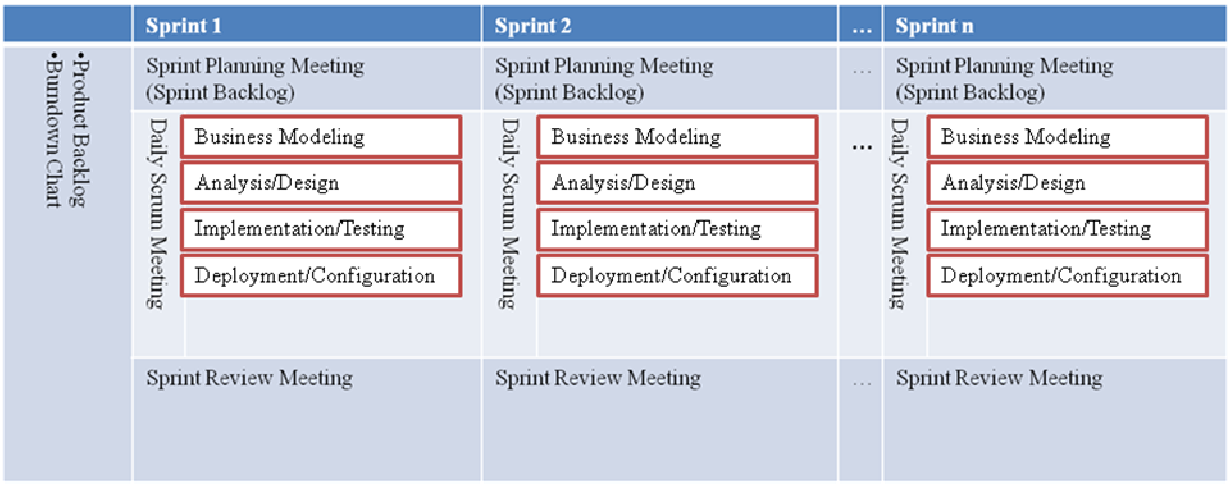



*Figure 16*. The inception phase of a hybrid model.

However, the hybrid model is capable of handling rapidly changing business requirements. It is expected that over-budget and delayed-schedule issues will be reduced due to the increased adaptability of the hybrid method. As explained, the four major phases and six disciplines of RUP provide a method platform, and the ceremonies, roles,

and artifacts of Scrum offer management and tracking mechanism in the new hybrid model.

### *Limitations of Present Study*

There are several limitations of this study. First, the most important limitation of this research is that the unit of analysis of this research was narrowed down to the Scrum software development process as utilized by the two study firms. Hence, the finding of this research may not be directly generalizable to the larger population and cannot be extended to wider populations. Second, interviewees were selected from various roles, levels of experience, and positions, including developers, lead software engineers, Scrum masters, project managers, and executive officers. Selection bias may have affected the process of selection. Third, critics may argue that the number of interview subjects is too small to be representative of the population. Interviews with a non-random sample of a few dozen members of the target population may not meet the statistical assumptions necessary to project the results accurately or reliably to the total population. Fourth, some interviewees might not want to present themselves negatively, and this research may not have recorded interviewees' actual opinions. Finally, because the quality of data collection and the research results is highly dependent on the skills of the researcher and on the rigor of data analysis, the quality of the research might be influenced by skills and experience of the researcher.

## *Conclusions*

This research identified the four main categories of critical issues and challenges that may affect the quality of the application of agile methods and illustrated a theoretical model which showed how agile methods can be adopted and utilized to effectively support the development of mission-critical, small- and large-scale projects. This paper also provided management guidelines to help organizations avoid and overcome obstacles in adopting the Scrum method as a future software development method. The lessons about Scrum obtained through the two case studies will be valuable assets to many Scrum practitioners, and the suggested new framework for further research on the application of traditional and agile methods will provides a basis for further research for those who want to further explore hybrid software development methods.

# REFERENCES


Advanced Development Methods, Inc. (2009). Scrum, Retrieved February 2, 2009, from http://www.controlchaos.com.

Agar, M. H. (1980). *The professional stranger: An informal introduction to ethnography*. New York: Academic Press.

AgileLogic. (2006). *Agile logic*. Retrieved March 20, 2009, from http://www.agilelogic.com.

Aiello, J. R., & Douthitt, E. A. (2001). Social facilitation from Triplett to electronic performance monitoring, *Group Dynamics*, *5*(3), 163-180.

Alavi, M., & Carlson, P. (1992). A review of MIS research and disciplinary development. *Journal of Management Information Systems, 8*(4), 45-62.

Ambler, S. (2005). A manager's introduction to the Rational Unified Process (RUP). Retrieved Feb 20, 2009, from http://www.ambysoft.com/downloads/managersIntroToRUP.pdf

Anacon, D. (1990). Outward bound: Strategies for team survival in an organization. *Academy of Management Journal, 33*(2), 334-365.

Avison, D., Lau, F., Myers, M., & Nielsen, P. A. (1999). Action research. *Communications of the ACM, 42*(1), 94-97.

Awad, M. A. (2005). *A comparison between agile and traditional software development methodologies*. Unpublished doctoral dissertation, The University of Western Australia, Australia.

Balijepally, V. (2005). Collaborative software development in agile methodologies – Perspectives from small group research. *Proceedings of the Eleventh Americas Conference on Information* Systems, August 11-14, Omaha, NE.

Baroudi, J. J., & Orlikowski, J. W. (1989). The problems of statistical power in MIS research. *MIS Quarterly,13*(1), 87-106.

Baskerville, R., & Myers, M. D. (2004). Special issues on action research in information systems: Making IS research relevant to practice – Forward. *MIS Quarterly, 28*(3), 329-335.

Baskerville, R.L., & Wood-Harper, A.T. (1996). A critical perspective on action research as a method for information systems research. *Journal of Information Technology, 11*, 235-246.

Batra, D., Sin, T., & Tseng, S. (2006). Modified agile practices for outsourced software projects. *Proceedings of the Twelfth Americas Conference on Information Systems,* Acapulco, Mexico, August 4-6, 2006, 3872-3880.

Beck, K. (2000). *Extreme programming explained: Embrace change*. Reading, MA: Addison-Wesley.

Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., et al. (2001). *Manifesto for agile software development.* Retrieved December 10, 2008, from http://www.agilemanifesto.org/

Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., et al. (2001). *Principles behind the agile manifesto.* Retrieved January 19, 2009, from http://www.agilemanifesto.org/principles.html

Benbasat, I., & McFarlan, W. (Eds.). (1984). *An analysis of research methodologies in the information systems research challenge.* Boston: Harvard Business School Press.

Benbasat, I., Goldstein, K., & Mead, M. (1987). The case research strategy in studies of information systems. *MIS Quarterly, 27*(2), 369-368.

Benbasat, I., & Zmud, W. R. (1999). Empirical research in information systems: The practice of relevance. *MIS Quarterly, 23*(1), 3-16.

Boehm, B. (2002, January). Get ready for agile methods with care. *Computer, 35*(1), 64-69.

Boehm, B., & Philip, P. (1988). Understanding and controlling software costs. *IEEE Transactions on Software Engineering, 14*(10), 1462-1477.

Boehm, B., & Turner, R. (2003, June). Using risk to balance agile and plan-driven methods. *Computer, 36*(6), 57-66.

Bond, C. F. (1982). Social facilitation: A self-presentational view. *Journal of Personality and Social Psychology,* 42, 1042-1050.

Bonoma, T. V. (1985). Case research in marketing: Opportunities, problems, and a process. *Journal of Marketing Research, 22*(2), 199-208.

Bowers, J., May, J., Melander, E., Baarman, M., & Ayoob, A. (2002). Tailoring XP for large systems mission critical software development. *Proceedings of the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods,* 100-111.

Brooks, F. P. (1995). *The mythical man-month.* Reading, MA: Addison-Wesley.

Cao, L., Mohan, K., Xu, P., & Ramesh, B. (2004). How extreme does extreme programming have to be? Adapting XP practices to large-scale projects. *Proceedings of the 37th Hawaii International Conference on System Sciences,* 1-10.

Charmaz, K. (2002). Qualitative interviewing and grounded theory analysis. In J. F. Gubrium & J.A. Holstein, *Handbook of qualitative research (2nd ed., pp.* 675-694). Thousand Oaks, CA: Sage.

Cockburn, A. (2007). *Agile software development: The cooperative game.* Upper Saddle River, NJ: Addison-Wesley.

Cottrell, N. B. (1972). Social facilitation, in C.G. McClintock (Ed.). *Experimental Social Psychology,* Holt, New York, 185-236.

Davies, L.J., & Nielsen, S. (1992). An ethnographic study of configuration management and documentation practices in an information technology centre. In K.E. Kendall, K. Lyytinen, & J. I. De Gross (Eds.), *The impact of computer supported technology on information systems development.*. Amsterdam: Elsevier/North Holland.

Dennis, A., Wixom, B. H., & Tegarden, D. (2005). *Systems analysis and design with UML version 2.0.* Hoboken, NJ: Wiley.

Denzin, N. K., & Lincoln, Y. S. (Eds.). (2000). *Handbooks of qualitative research* (2nd ed.). Thousand Oaks, CA: Sage.

Eisenhardt, K. M. (1989). Building theories from case study research. *Academy of management review, 14*(4), 532-550.

Elsbach, K. D., & Sutton, R. I. (1992). Acquiring organizational legitimacy through illegimate actions: A marriage of institutional and impression management theories. *Academy of Management Journal, 35*(4), 699-733.

Erickson, F. (1973). What makes school ethnography 'ethnography'? *Council on Anthropology and Education Newsletter, 4*(2), 10-19.

Forrester Research, Inc. (2005). *Corporate IT leads the second wave of agile adoption.* Cambridge, MA.

Fowler, M. (2005). *The new methodology.* Retrieved December 10, 2008, from http://martinfowler.com/articles/newMethodology.html

Fruhling, A., & De Vreede, G. J. (2006). Field experiences with extreme programming: Developing an emergency response system. *Journal of Management Information Systems, 22*(4), 39-68.

Gall, D. M., Gall, P. J., & Borg, R. W. (2003). *Educational research: An introduction.* Boston: Allyn and Bacon.

Glaser, B. G. (1978). *Theoretical sensitivity: Advances in the methodology of grounded theory.* Mill Valley; CA: Sociology Press.

Glaser, B. G., & Strauss, A. L. (1967). *The discovery of grounded theory: Strategies for qualitative research*. New York: Aldine.

Glesne, C. (2006). *Becoming qualitative researchers.* Boston: Pearson.

Hickey, A. M., & Davis, A. M. (2004). A unified model of requirements elicitation. *Journal of Management Information Systems, 20*(4), 65-84.

Highsmith, J., & Cockburn, A. (2001, September). Agile software development: The business innovation. *IEEE Computer, 34*(9), 120-122.

Hodgetts, P. (2009). *Product development with Scrum*. Retrieved February 1, 2009, from http://www.agilelogic.com.

Isabella, L. A. (1990). Evolving interpretations as a change unfolds: How managers construe key organizational events. *Academy of Management Journal, 33*(1), 7-41.

Jones, C. (1997). *Applied Software Measurements.* Highstown, NJ: McGraw-Hill.

Kahn, W. A. (1990). Psychological conditions of personal engagement and disengagement at work. *Academy of Management Journal, 33*(4), 248-266.

Kaplan, B., & Maxwell, J. A. (1984). Qualitative research methods for evaluating computer information systems. Thousand Oaks, CA: Sage.

Kaplan, R. S. (1985). *The role of empirical research in management accounting*. Boston: Harvard Business School.

Kruchten, P. (2004). *The rational unified process: An introduction (*3rd ed.*).* Reading, MA: Addison-Wesley Longman.

Larman, C. (2007). *Agile and iterative development*. Boston: Addison-Wesley.

Leffingwell, D. (2007). *Scaling software agility: Best practices for large enterprises.* Upper Saddle River, NJ: Addison-Wesley.

Lévi-Strauss, C. (1966). *The savage mind* (2nd ed.). Chicago: University of Chicago Press.

Lindvall, M., Muthig, D., Dagnino, A., Wallin, C., Stupperich, M., Kiefer, D., et al. (2004). Agile software development in large organizations. *Computer, 37*(12), 26-34.

Lippert, M., Becker-Pechau, P., Breitling, H., Koch, J., Kornstadt, A., Roock, S., et al. (2003). Developing complex projects using XP with extensions. *Computer, 36*(6), 67-73.

Leonard-Barton, D. A. (1990). A dual methodology for case studies: Synergistic use of a longitudinal single site with replicated multiple sites. *Organization Science, 1*(3), 248-266.

Markus, M. L. (1983). Power, politics, and MIS implementation. *Communications of the ACM, 26*(6), 430-444.

Markus, M. L., & Lee, S. A. (1999). Special issues on intensive research in information systems: Using qualitative, interpretive, and case methods to study information technology – forward. *MIS Quarterly, 23*(1), 35-38.

Marrington, A., Hogan, J. M., & Thomas, R. (2005). Quality assurance in a student-based agile software engineering process. *Proceedings of the 2005 Australian Software Engineering Conference,* 324-331.

Martin, P. Y., & Turner, B. A. (1986). Grounded theory and organizational research. *The Journal of Applied Behavioral Science*, *22*(2), 141-157.

Miller, K., & Larson, D. (2005, winter). Agile software development: Human values and culture. *Technology and Society Magazine, IEEE, 24*(4), 36-42.

Myers, M. D. (1997). Qualitative research in information systems. *MIS Quarterly*, *21*(2), 241-242.

Myers, M. D. (1999). Investigating information systems with ethnographic research. *Communication of the AIS, 2*(23), 1-20.

Myers, M. D. (2009). *Qualitative research in business and management*. London: Sage.

Nelson, C., Treichler, P. A., & Grossberg, L. (1992). *Cultural studies: An introduction.* New York: Routledge.

Orlikowski, W. J. (1991). Integrated information environment or matrix of control? The contradictory implications of information Technology. *Accounting, Management and Information Technologies, 1*(1), 9-42.

Orlikowski, W. J. (1993). CASE tools are organizational change: Investigating incremental and radical changes in systems development. *MIS Quarterly*, *17*(3), 309-340.

Orlikowski, W. J., & Baroudi, J. J. (1991). Studying information technology in organizations: Research approaches and assumptions", *Information Systems Research, 2*(1), 1-28.

Parnas, D. (2006). Agile methods and GSD: The wrong solution to an old but real problem. *Communication of the ACM, 49*(10), 29.

Parrish, A., Smith, R., Hale, D., & Hale, J. (2004). A field study of developer pairs: Productivity impacts and implications. *IEEE Software, 21*(5), 76-79.

Pettigrew, A. M. (1989). Issues of time and site selection in longitudinal research on change. In J. I. Cash, & P. R. Lawrence (Eds.). *The information systems research challenge: Qualitative research methods*, (pp. 13-19). Boston: Harvard Business School Press.

Pettigrew, A. M. (1990). Longitudinal field research on change: Theory and practice. *Organization Scrience, 1*(3), 267-292.

Poppendieck, M., & Poppendieck, T. (2003). *Lean software development. An agile toolkit.* Upper Saddle River, NJ: Addison-Wesley.

Preston, A. M. (1991). The 'Problem' in and of management information systems, *Accounting, Management and Information Technologies*, *1*(1), 43-69.

Royce, W. (1970). Managing the development of large software systems. *Proceedings of IEEE WESCON,* 1-9.

Satzinger, J. W., Jackson, R. B., & Burd, S. D. (2005). *Object-oriented analysis & design with unified process.* Boston: Thomson Course- Technology.

Schach, S. R. (2004). *An introduction to object-oriented systems analysis and design with UML and the unified process*. Boston: McGraw-Hill.

Schman, L. (1987). *Plans and situated actions: The problem of human-machine communication.* Cambridge: Cambridge University Press.

Schwaber, K. (1996). SCRUM development process. *Proceedings of ACM SIGPLAN on Objected-Oriented Programming, Systems, Languages, & Applications* (OOPSLA ’96), San Jose, CA.

Schwaber, K. (2004). *Agile project management with Scrum.* Redmond, WA: Microsoft Press.

Schwaber, K. (2007). *The enterprise and Scrum.* Redmond, WA: Microsoft Press.

Schwaber, K. (2008). *What is Scrum?* Retrieved March 5, 2008, from http://www.scrumalliance.org/system/resource/file/275/whatIsScrum.pdf.

Schwaber, K., & Beedle, M. (2002). *Agile software development with Scrum*, Upper Saddle River, NJ: Prentice Hall.

Shine Technologies. (2003). *Agile methodologies survey results*. Vitoria, Australia.

Shingo, S. (1981). S*study of 'Toyota' production system from industrial engineering viewpoint.* Osaka, Japan: Japan Management Association.

Simon, M. (2006). Global software development: A hard problem requiring a host of solutions. *Communication of the ACM, 49*(10), 32-33.

Sommerville, I. (2004). *Software Engineering.* Boston: Addision-Wesley.

Spencer, J. (2005). There has to be a better way! *Proceedings of Agile United, Experience Report.*

Standish Group International. (1994). *The Chaos report.* Retrieved March 20, 2007, from http://www.standishgroup.com/sample_research/chaos_1994_1.php

Standish Group International. (2001). *Extreme Chaos,* Retrieved July 10, 2009, from http://www.smallfootprint.com/LinkClick.aspx?fileticket=XJyTlOiQ1Zw%3D&tabid=63&mid=519

Stone, E. (1978). *Research methods in organizational behavior*. Glenview, IL: Scott, Foresman.

Straus, A. L., & Corbin, J. M. (1988). Basics of qualitative research: Techniques and procedures for developing grounded theory (2nd ed.). Thousand Oaks, CA: Sage.

Straus, A L., & Coribin, J. M. (1990). *Basics of qualitative research: Grounded theory, procedures, and techniques.* Newbury Park, CA: Sage.

Suchman, L. (1955). Making work visible. *Communications of the ACM*, *38*(9), 56-64.

Suchman, L. (1987). *Plans and situated actions: The problem of human-machine communication.* Cambridge: Cambridge University Press.

Sutton, R. I. (1987). The process of organizational death: Disbanding and reconnecting. *Administrative Science Quarterly, 32*(4), 542-569.

Takeuchi, H., & Nonaka, I. (1986, January- February). The new new product development game. *Harvard Business Review*, 137-146.

Thomas, M. (2001). It projects sink or swim. *British Computer Society Review*.

Turner, B. A. (1983). The use of grounded theory for the qualitative analysis of organizational behavior. *Journal of Management Studies, 20*(34), 333-348.

Van Maanen, J. (1982). *Introduction in varieties of qualitative research.* Beverly Hills, CA: Sage.

Watson, R. T., Kelly, G., Galliers, D., & Brancheau, C. (1997). Key issues in information systems management: An international perspective. *Journal of Management Information Systems, 13*(4), 91-115.

Weick, K. (1994). Theoretical assumptions and research methodology selection. Boston: Harvard Business Press.

Williams, L., & Cockburn, A. (2003, June). Agile software development: It's about feedback and change. *Computer, 36*(6), 39-43.

Williams, K. D., Harkins, S. G., & Karau, S. J. (1991). Social loafing and social compensation: The effects of expectation of co-worker performance. *Journal of Personality and Psychology, 61*(4), 570-581.

Williams, L., & Kessler, R. (2000). All I really need to know about pair programming I learned in kindergarten. *Communication of the ACM, 43*(5), 108-114.

Willson, C. D. (2009). *A brief introduction to SCRUM: An agile methodology*. Matincor Inc. Information Technology Management Consulting.

Wynn, E. (1979). *Office conversation as an information medium*. Unpublished doctoral dissertation, University of California, Berkeley.

Yin, R. K. (1989a). *Case study research: Design and methods*. Beverly Hills, CA: Sage.

Yin, R. K. (1989b). Research design issues in using the case study method to study management information systems. In J. I. Cash, Jr. & P. R. Lawrence (Eds.), *The information systems research challenge: Qualitative research methods.* Boston: Harvard Business School Press.

Yin, R. K. (1993). *Applications of case study research*. Newbury Park, CA: Sage.

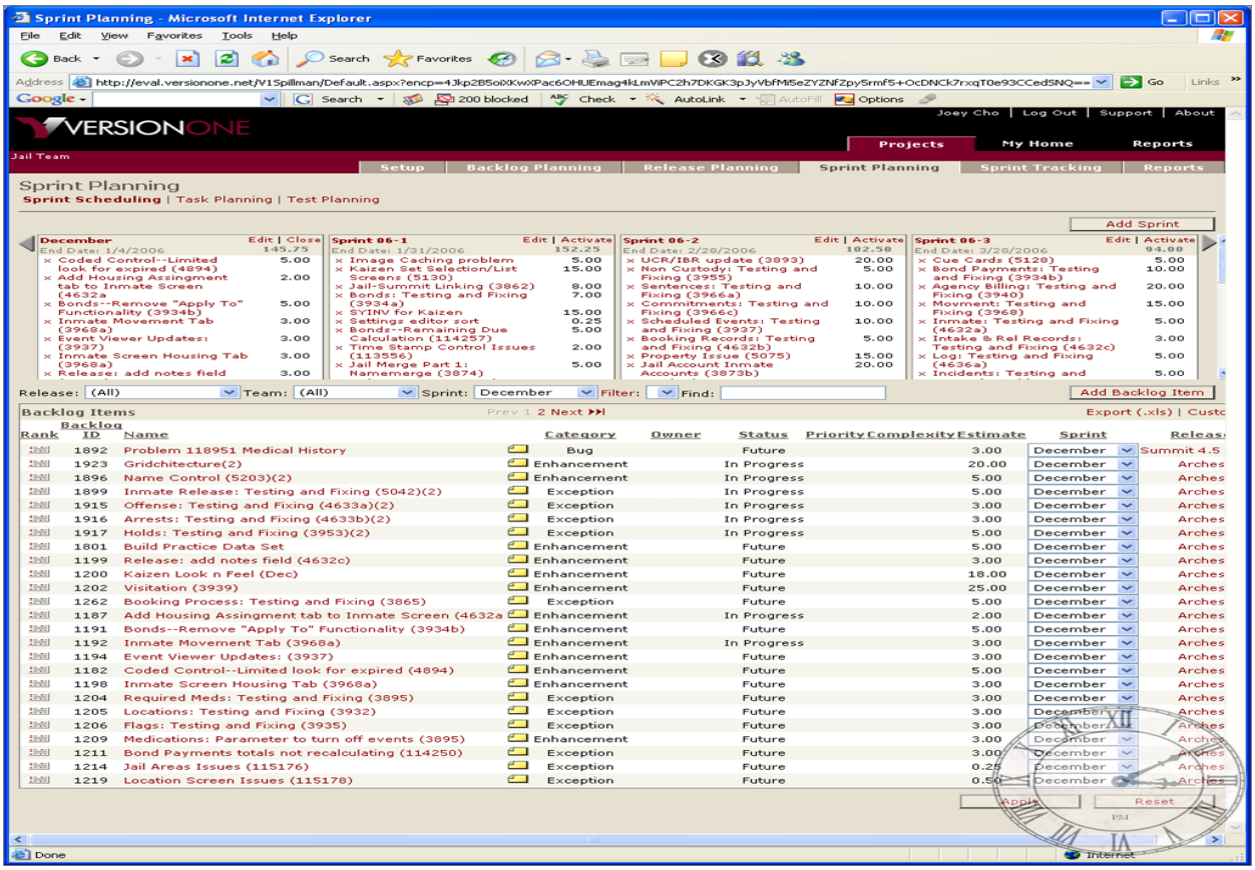
Yin, R. K. (2003). *Case study research: Design and methods* (3rd *ed*.). Thousand Oaks, CA: Sage.

Ytterstad, P., Akselsen, S., Svendsen, G., & Watson, R.T. (1996). Teledemocracy: Using information technology to enhance political work, *MISQ Discovery, 1*.

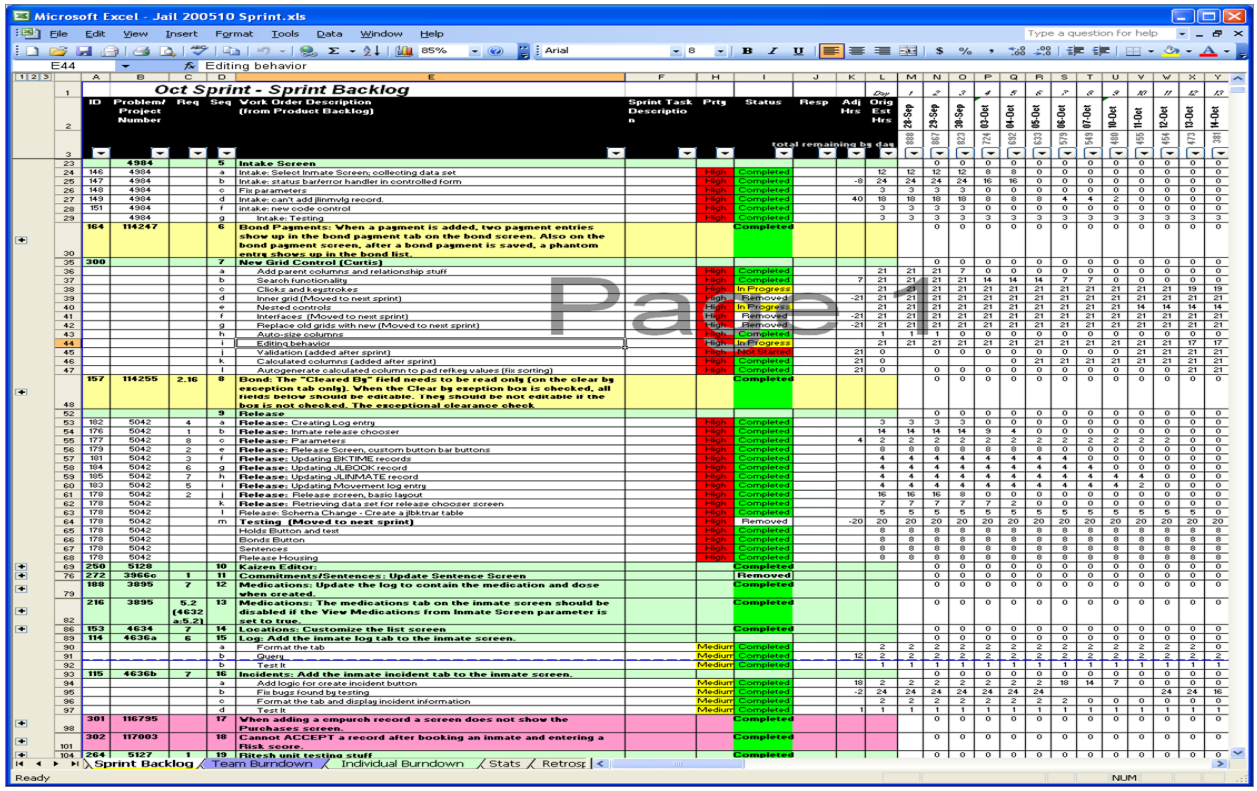
Zuboff, S. (1988). *In the age of the smart machine*. New York: Basic Books.

# APPENDICES

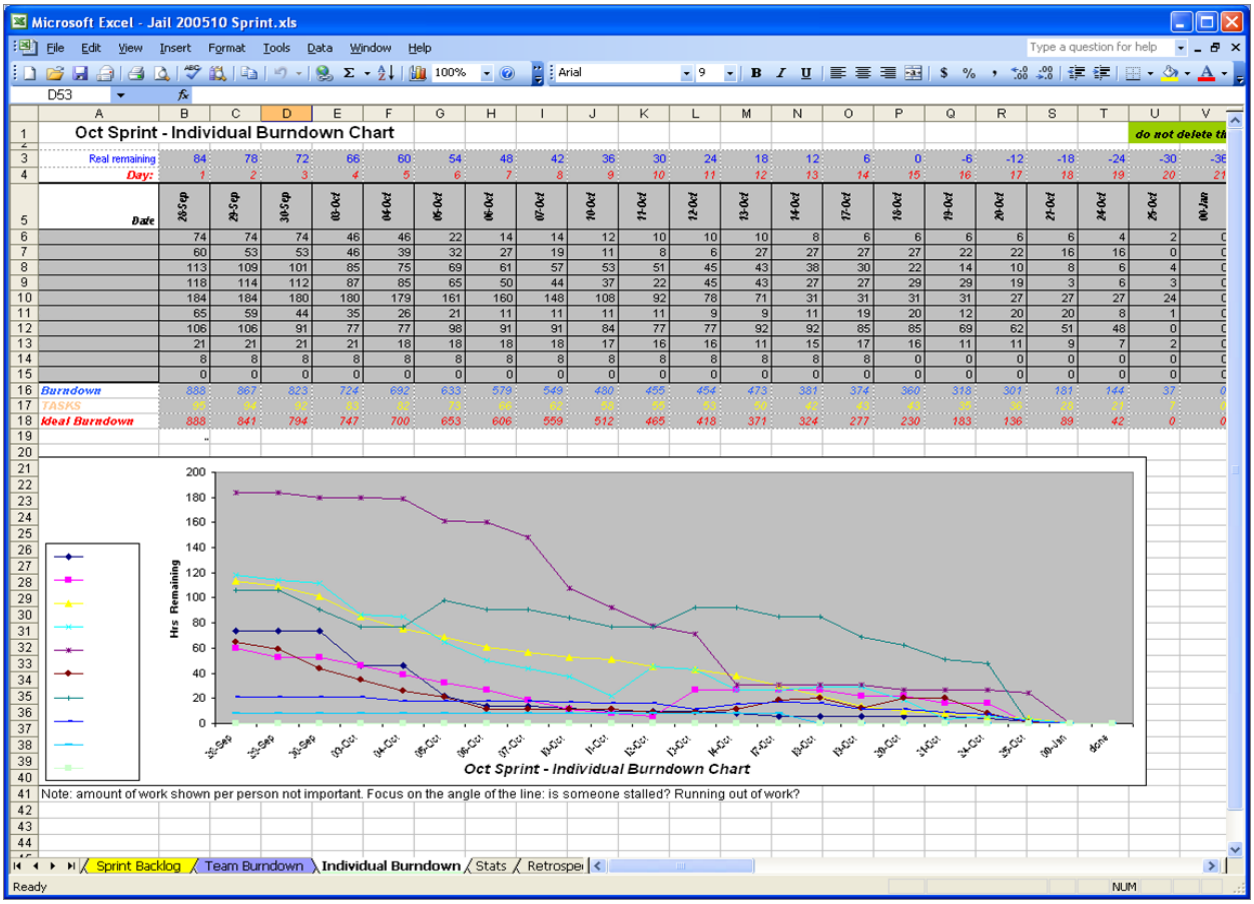
# APPENDIX A. PRODUCT BACKLOG


Sprint Planning - Microsoft Internet Explorer
VERSIONONE
Projects
My Home
Reports
Setup
Backlog Planning
Release Planning
Sprint Planning
Sprint Tracking
Reports
Sprint Planning
Sprint Scheduling | Task Planning | Test Planning
Add Sprint
December
Sprint 06-1
Sprint 06-2
Sprint 06-3
Backlog Items
Add Backlog Item
Export (.xls)

# APPENDIX B. SPRINT BACKLOG


Microsoft Excel - Jail 200510 Sprint.xls
File Edit View Insert Format Tools Data Window Help
Oct Sprint - Sprint Backlog
Sprint Backlog
Team Burndown
Individual Burndown
Stats
Retrosp
Ready

# APPENDIX C. BURNDOWN CHART


Microsoft Excel - Jail 200510 Sprint.xls
Oct Sprint - Individual Burndown Chart
Hrs Remaining
Oct Sprint - Individual Burndown Chart
Sprint Backlog
Team Burndown
Individual Burndown
Stats
Retrospe
Ready
NUM

# APPENDIX D. C# CODING STANDARD

# C# Coding Standards

### *Introduction*

The following is a paragraph from a Microsoft design guideline found in Microsoft Visual C# .NET.

"The .NET Framework's managed environment allows developers to improve their programming model to support a wide range of functionality. The goal of the .NET Framework design guidelines is to encourage consistency and predictability in public APIs while enabling Web and cross-language integration. It is strongly recommended that you follow these design guidelines when developing classes and components that extend the .NET Framework. Inconsistent design adversely affects developer productivity. Development tools and add-ins can turn some of these guidelines into de facto prescriptive rules, and reduce the value of nonconforming components. Nonconforming components will function, but not to their full potential."

Though this statement refers specifically to library development, it applies just as well to all our C# development. Our goal is to create code that is easily maintainable, expandable and conforming to standards that help us interoperate with other systems, code pieces or applications.

### *File, Class and Method headers*

#### File Headers

The copyright notice and the CVS information are all that will appear at the top of a .cs file. These, along with top and bottom separator bars, will account for the first 4 lines of each file.

```
//===============================================
// Copyright 2004 ABC Technologies Inc. All Rights Reserved.
// Version $Id: $
//===============================================
```

#### Class/Delegate/Interface Headers

Class headers will be in xml format to facilitate the automatic documentation feature. Some of this header may be entered automatically if you are using Visual Studio. This is the general format:

```
/// <summary>
/// Short description of the Class
/// </summary>
public class MyClass
{…
```

If the class description is longer than one line, add a remarks section. The remarks section may be as many lines as needed to fully describe the class. Here is an example:

```
/// <remarks>
/// This is a full description of the class ...
/// ...
/// </remarks>
```

## Method/Event Handler Headers

Function headers will be in xml format to facilitate the automatic documentation feature. Some of this header may be entered automatically if you are using Visual Studio. This is the general format:

```
/// <summary>
/// Short description of the Class
/// </summary>
/// <param name="row">Row we're looking for</param>
/// <returns>description of return value</returns>
public bool rowExists( DataRow row )
{
     …
```

If there is a need for a description longer than one line, use the remarks tag as described above, in the File Headers section.

Exceptions - if there are any exceptions thrown in the function, they **must** be documented inside the exception tag in the function header:

```
/// <exception> exception name </exception>
```

## Properties Headers

Class properties should be documented in the following manner:

```
/// <summary>
/// Name property
/// </summary>
/// <value>
/// A value tag is used to describe the property value
/// </value>
public string Name
{
     get
     {   …
```

A remarks tag (as described above) may be used for more detailed explanations, but generally will not be found in properties.

## Documenting Code

The following points are recommended commenting techniques.

- When modifying code, always keep the commenting around it up to date.
- Avoid adding comments at the end of a line of code; end-line comments make code more difficult to read. However, end-line comments are appropriate when annotating variable declarations, in which case, align all end-line comments at a common tab stop.
- Avoid clutter comments, such as an entire line of asterisks. Instead, use white space to separate comments from code.
- Prior to deployment, remove all temporary or extraneous comments to avoid confusion during future maintenance work.
- If you need comments to explain a complex section of code that you are writing, examine the code to determine if you should rewrite it. If at all possible, do not document bad code — rewrite it. Although performance should not typically be sacrificed to make the code simpler for human consumption, a balance must be maintained between performance and maintainability.
- Use complete sentences when writing comments. Comments should clarify the code, not add ambiguity.
- Comment as you code because you will not likely have time to do it later. Also, should you get a chance to revisit code you have written, that which is obvious today probably will not be obvious six weeks from now.
- Use comments to explain the intent of the code. They should not serve as inline translations of the code.
- Comment anything that is not readily obvious in the code.
- Use comments on code that consists of loops and logic branches. These are key areas that will assist source code readers.
- Throughout the application, construct comments using a uniform style with consistent punctuation and structure.
- Separate comments from comment delimiters with white space. Doing so will make comments obvious and easy to locate when viewed without color clues.

### *Naming conventions*

## Capitalization Rules

The following table summarizes the capitalization rules and provides examples for the different types of identifiers.

| Identifier | Case | Example |
|---|---|---|
| Class | Pascal | **AppDomain** |
| Enum type | Pascal | **ErrorLevel** |
| Enum values | Pascal | **FatalError** |
| Event | Pascal | **ValueChange** |
| Exception class | Pascal | **WebException**<br>**Note** Always ends with the suffix **Exception.** |
| Read-only Static field | Pascal | **RedValue** |
| Interface | Pascal | **IDisposable**<br>**Note** Always begins with the prefix I. |
| Method | Pascal | **ToString** |
| Namespace | Pascal | **System.Drawing** |
| Parameter | Camel | **typeName** |
| Variable | Camel | **fourLeafCIover** |
| Property | Pascal | **BackColor** |
| Protected instance field | Camel | **redValue**<br>**Note** Rarely used. A property is preferable to using a protected instance field. |
| Public instance Field | Pascal | **RedValue**<br>**Note** Rarely used. A property is preferable to using a public instance field. |

## Name Guidelines

The following points are recommended naming techniques.

### *Routines*

- Avoid elusive names that are open to subjective interpretation, such as `AnalyzeThis()` for a routine, or `xxK8` for a variable. Such names contribute to ambiguity more than abstraction.
- In object-oriented languages, it is redundant to include class names in the name of class properties, such as `Book.BookTitle`. Instead, use `Book.Title`.
- Use the verb-noun method for naming routines that perform some operation on a given object, such as `CalculatelnvoiceTotal ()`.

### *Variables*

- Append computation qualifiers (`Avg, Sum, Min, Max, Index`) to the end of a variable name where appropriate.
- Use complementary pairs in variable names, such as min/max, begin/end, and open/close.

- Since most names are constructed by concatenating several words, use mixed-case formatting to simplify reading them. In addition, to help distinguish between variables and routines, use Pascal casing (`CalculatelnvoiceTotal`) for routine names where the first letter of each word is capitalized. For variable names, use camel casing (`documentFormatType`) where the first letter of each word except the first is capitalized.
- Even for a short-lived variable that may appear in only a few lines of code, still use a meaningful name. Use single-letter variable names, such as `i`, or `j`, for short-loop indexes only.
- Do not use literal numbers (magic numbers) or literal strings, such as `For i = l To 7`. Instead, use named constants, such as `For i = 1 To NUM_DAYS_IN_WEEK` for ease of maintenance and understanding.

### *Miscellaneous*

- Minimize the use of abbreviations, but use those that you have created consistently. An abbreviation should have only one meaning and likewise, each abbreviated word should have only one abbreviation. For example, if you use min to abbreviate minimum, do so everywhere and do not use min to also abbreviate minute.
- When naming functions, include a description of the value being returned, such as `GetCurrentWindowName{).`
- Avoid reusing names for different elements, such as a routine called `ProcessSales()`and a variable `called iProcessSales.`
- Avoid homonyms, such as write and right, when naming elements to prevent confusion during code reviews.

### *Abbreviations*

To avoid confusion and guarantee cross-language interoperation, follow these rules regarding the use of abbreviations:

- Do not use abbreviations or contractions as parts of identifier names. For example, use `GetWindow` instead of `GetWin`.
- Do not use acronyms that are not generally accepted in the computing field.
- Where appropriate, use well-known acronyms to replace lengthy phrase names. For example, use `UI` for User Interface and `OLAP` for On-line Analytical Processing.
- When using acronyms, use Pascal case or camel case for acronyms more than two characters long. For example, use HtmlButton or htmlButton. However, you should capitalize acronyms that consist of only two characters, such as `System. IO` instead of `System. Io.`
- Do not use abbreviations in identifiers or parameter names. If you must use abbreviations, use camel case for abbreviations that consist of more than two characters, even if this contradicts the standard abbreviation of the word.

## Avoiding Type Name Confusion

Different programming languages use different terms to identify the fundamental managed types. Class library designers must avoid using language-specific terminology. Follow the rules described in this section to avoid type name confusion.

Use names that describe a type's meaning rather than names that describe the type. In the rare case that a parameter has no semantic meaning beyond its type, use a generic name. For example, a class that supports writing a variety of data types into a stream might have the following methods.

```
void Write{double value);
```

Do not create language-specific method names, as in the following example.

```
void Write(double doubleValue);
```

## Case Sensitivity

To avoid confusion and guarantee cross-language interoperation, follow these rules regarding the use of case sensitivity:

- Do not use names that require case sensitivity. Components must be fully usable from both case-sensitive and case-insensitive languages. Case-insensitive languages cannot distinguish between two names within the same context that differ only by case. Therefore, you must avoid this situation in the components or classes that you create.
- Do not create two namespaces with names that differ only by case. For example, a case insensitive language cannot distinguish between the following two namespace declarations.

  ```
  namespace ee.cummings;
  namespace Ee.Cummings;
  ```
- Do not create a function with parameter names that differ only by case. The following example is incorrect.

  ```
  void MyFunction(string a, string A)
  ```
- Do not create a namespace with type names that differ only by case. In the following example, `Point p` and `POINT p` are inappropriate type names
  because they differ only by case.

  ```
  System.Windows.Forms.Point p
  System.Windows.Forms.POINT p
  ```
- Do not create a type with property names that differ only by case. In the following example, `int Color` and `int COLOR` are inappropriate property names
  because they differ only by case.

  ```
  int Color {get, set}
  int COLOR {get, set}
  ```

- Do not create a type with method names that differ only by case. In the following example, `calculate` and `Calculate` are inappropriate method names because they differ only by case.

```
void calculate()
void Calculate()
```

## Namespace Naming Guidelines

The general rule for naming namespaces is to use the company name followed by the technology name and optionally the feature and design as follows.

CompanyName.TechnologyName[.Feature][.Design]

For example:

ABC.
Microsoft.Media.Design

- Prefixing namespace names with a company name or other well-established brand avoids the possibility of two published namespaces having the same name. For example, `Microsoft.Office` is an appropriate prefix for the Office Automation Classes provided by Microsoft.
- Use a stable, recognized technology name at the second level of a hierarchical name. Use organizational hierarchies as the basis for namespace hierarchies. Name a namespace that contains types that provide design-time functionality for a base namespace with the `.Design` suffix. For example, the svstem.windows.Forms.Desian Namespace contains designers and related classes used to design Svstem.windows.Forms based applications.
- A nested namespace should have a dependency on types in the containing namespace. For example, the classes in the Svstem.Web.UI.Design depend on the classes in Svstem.Web.UI. However, the classes in **System.Web.UI** do not depend on the classes in **System.Web.UI.Design.**
- You should use Pascal case for namespaces, and separate logical components with periods, as in `Microsoft.Office.PowerPoint`. If your brand employs nontraditional casing, follow the casing defined by your brand, even if it deviates from the prescribed Pascal case. For example, the namespaces `NeXT.Webobjects` and `ee.cummings` illustrate appropriate deviations from the Pascal case rule.
- Use plural namespace names if it is semantically appropriate. For example, use `System.Collections` rather than `System.Collection`. Exceptions to this rule are brand names and abbreviations. For example, use `System.IO` rather than `System.IOs.`
- Do not use the same name for a namespace and a class. For example, do not provide both a `Debug` namespace and a `Debug` class.
- Finally, note that a namespace name does not have to parallel an assembly name. For example, if you name an assembly `MyCompany.MyTechnology.dll`, it

does not have to contain a `MyCompany.MyTechnoiogy` namespace.

## Additional Class Naming Guidelines

The following rules outline the guidelines for naming classes:

- Use a noun or noun phrase to name a class.
- Use Pascal case.
- Use abbreviations sparingly.
- Do not use a type prefix, such as `C` for class, on a class name. For example, use the class name `Filestream` rather than `CFileStream`.
- Do not use the underscore character (_).
- Occasionally, it is necessary to provide a class name that begins with the letter I, even though the class is not an interface. This is appropriate as long as I is the first letter of an entire word that is a part of the class name. For example, the class name `Identitystore` is appropriate.
- Where appropriate, use a compound word to name a derived class. The second part of the derived class's name should be the name of the base class. For example, `ApplicationException` is an appropriate name for a class derived from a class named `Exception`, because `ApplicationException` is a kind of `Exception`. Use reasonable judgment in applying this rule. For example, `Button` is an appropriate name for a class derived from `Control`. Although a button is a kind of control, making `Control` a part of the class name would lengthen the name unnecessarily.

## Additional Parameter Naming Guidelines

It is important to carefully follow these parameter naming guidelines because visual design tools that provide context sensitive help and class browsing functionality display method parameter names to users in the designer. The following rules outline the naming guidelines for parameters:

- Use camel case for parameter names.
- Use descriptive parameter names. Parameter names should be descriptive enough that the name of the parameter and its type can be used to determine its meaning in most scenarios. For example, visual design tools that provide context sensitive help display method parameters to the developer as they type. The parameter names should be descriptive enough in this scenario to allow the developer to supply the correct parameters.
- Use names that describe a parameter's meaning rather than names that describe a parameter's type. Development tools should provide meaningful information about a parameter's type. Therefore, a parameter's name can be put to better use by describing meaning. Use type-based parameter names sparingly and only where it is appropriate.
- Do not use reserved parameters. Reserved parameters are private parameters that might be exposed in a future version if they are needed. Instead, if more data is needed in a future version of your class library, add a new overload for a method.

- Do not prefix parameter names with Hungarian type notation.

### *Format*

Formatting makes the logical organization of the code obvious. Taking the time to ensure that the source code is formatted in a consistent, logical manner is helpful to you and to other developers who must decipher the source code.

The following points are recommended formatting techniques.

#### Braces

- Align open and close braces vertically where brace pairs align, such as:

```
for( i  = 0; i < 100; i++
) {
    …
}
else
{
    …
}
```

  This is the default behavior of Microsoft tools and the common coding style for all public C# files.
- Braces are required on all constructs that allow the possibility of braces.

#### White Space

- Use tabs for indenting lines. This allows different users to set tabs in whatever manner they choose.
- Indent code along the lines of logical construction. Without indenting, code becomes difficult to follow. Indenting the code yields easier-to-read code, such as:

```
If ... Then
   If   ... Then
            ...
   Else
            ...
   End
If Else
```

```
   ...
End If
```

- Lines of code and comments shall not exceed 100 columns. Lines that are longer than 100 columns should be extended to a new line at a reasonable, readable location.

- Use spaces before and after most operators when doing so does not alter the intent of the code.
  For example: `x = y * x;` instead of `x=y*x;`
- Use white space to provide organizational clues to source code. Doing so creates "paragraphs" of code, which aid the reader in comprehending the logical segmenting of the software.
- When a line is broken across several lines, make it obvious that it is incomplete without the following line by placing the concatenation operator at the end of each line instead of at the beginning.
- Where appropriate, avoid placing more than one statement per line. An exception is a for loop, such as `for ( i = 0; i < 100; i++ ).`

## Horizontal Spacing

Horizontal spacing for keywords/methods followed by a parenthesis use the following format. The engineer can use their own discretion for adding a space between the keyword/method name and the opening parenthesis and between the closing parenthesis and the opening brace. The coding standard does require a space to be present after an open parenthesis and a before a closing parenthesis. Any expression between the parenthesis follows the above rule for horizontal space.

```
Correct Examples:
classMethod( oneParameter );
classMethod ( oneParameter );
if( ( screenWidth <= 80 ) && ( screenHeight > 20 ) ){
if ( ( screenWidth <= 80 ) && ( screenHeight > 20 ) ) {

Incorrect Examples:
classMethod(oneParameter);
if((screenWidth<=80)&&(screenHeight>20)){
```

## Modules

Break large, complex sections of code into smaller, comprehensible modules.

## Exception Handling

- Developers are responsible for what do in response to exceptions.

- In case of exceptions, give a friendly message to the user only if:
    - The message can be understood by the user.
    - The message conveys useful information to the user.
- Do not write try-catch in all your methods. Use it only if there is a possibility that a specific exception may occur. For example, if you are writing into a file, handle only FileIOException.

- You may write your own custom exception classes, if required in your application. Do not derive your custom exceptions from the base class SystemException. Instead, inherit from ApplicationException.

## *Commenting*

C# provides a mechanism for developers to document their code using XML. This document contains the standard keywords and formats we will use to ensure the help files are universally useful.

In source code files, lines that begin with /// and that precede a user-defined type such as a class, delegate, or interface; a member such as a field, event, property, or method; or a namespace declaration can be processed as comments and placed in a file.

Items that are documented in this fashion are

- Classes
- Delegates
- Interfaces
- Members
    - Field
    - Property
    - Event
    - Method

The summary tag is the most basic of tags. The list below is the complete set currently supported by VS.NET. The ones marked with a * are the ones I feel are the most useful.

- The c tag gives you a way to indicate that text within a description should be marked as code. Use code to indicate multiple lines as code.
- code*
  The code tag gives you a way to indicate multiple lines as code. Use <c> to indicate that text within a description should be marked as code.
- example*
  The example tag lets you specify an example of how to use a method or other library member. Commonly, this would involve use of the code tag.
- exception*
  The exception tag lets you specify which exceptions a class can throw.

- include
  The include tag lets you refer to comments in another file that describe the types and members in your source code. This is an alternative to placing documentation comments directly in your source code file.
- para
  The para tag is for use inside a tag, such as <remarks> or <returns>, and lets you add structure to the text.

- param*
  The param tag should be used in the comment for a method declaration to describe one of the parameters for the method.
- paramref
  The paramref tag gives you a way to indicate that a word is a parameter. The XML file can be processed to format this parameter in some distinct way.
- permission*
  The permission tag lets you document the access of a member. The System.Security.PermissionSet lets you specify access to a member.
- remarks*
  The remarks tag is where you can specify overview information about a class or other type. <summary> is where you can describe the members of the type.
- returns
  The returns tag should be used in the comment for a method declaration to describe the return value.
- see
  The see tag lets you specify a link from within text. Use <seealso> to indicate text that you might want to appear in a See Also section.
- seealso*
  The seealso tag lets you specify the text that you might want to appear in a See Also section. Use <see> to specify a link from within text.
- summary*
  The summary tag should be used to describe a member for a type. Use <remarks> to supply information about the type itself.
- value*
  The value tag lets you describe a property. Note that when you add a property via code wizard in the Visual Studio .NET development environment, it will add a <summary> tag for the new property. You should then manually add a <value> tag to describe the value that the property represents.

Here are my suggestions for what to include for each of the types we will be commenting:

### *Classes, Delegates and Interfaces*

I suggest we include the summary and optionally, remarks to further clarify or provide more detail of the type's behavior.

Required: summary Optional: remarks
What about: permission?

```
/// <summary>
/// This is a short description of the class Sample
/// </summary>
/// <remarks>If more needs to be said about this class,
place those comments
/// in the remarks section.

/// </remarks>
public class Sample
{
```

We may also want to require the permission flag to indicate the visibility of the type. Do you feel this would be valuable to have this show in the documentation?

### *Member fields*

Required: summary
Optional: remarks
Question: permission?

```
public class Sample
{
     /// <summary>
     /// short description of value
     /// </summary>
     /// <permission>private</permission>
     private int value;
```

### *Member properties*

Required: summary
value
Optional: remarks

```
/// <summary>
/// Name property
/// </summary>
/// <value>
/// A value tag is used to describe the property value
/// </value>
public string Name
{
     get
```

```
    {
```

### *Member methods and event handlers*

Required: summary
returns
params (only if present)
exceptions (only if any are thrown)

Optional: remarks
Question: permission?

```
/// <summary>
/// Short description
/// </summary>
/// <param name="sender">description of sender</param>
/// <param name="e">description of e</param>
/// <returns>void</returns>
private void toolBar_ButtonClick(object sender,
System.Windows.Forms.ToolBarButtonClickEventArgs e)
    {
```

These are only my suggestions. Any of this is up for debate, so please share your opinions. I suggest we require the returns field, but let it be known if you disagree.

What do you think about having the permission included? Necessary? Not enough need?

And what about formatting? There are 3 different types shown here - do you have a preference? I use the one line method when the lines are short. But on multi-line entries, which of the 2 outlined above work for you?

Since .Net was release, MS added the ability to use /** */ to surround multi-line text. This is not put in automatically, so I suggest that for now we use the /// method. But maybe you disagree??

Note: We are able to define any tags we want as the output is XML. But in order to take advantage of some cool features, I suggest we use the predefined tags.

# APPENDIX E. INTERVIEW QUESTIONS

# Interview Questions for Developers and Quality Assurance Personnel

**Team**

- How many teams are involved in the project?
- When a team is composed, what kinds of characteristics (programming skills, knowledge, experiences, etc.) of developers are considered?
- How many people are working on the project in each team? (specify by title)
    - Software Developer:
    - Quality Assurance Personnel:
    - Product Manager:
- Who else was on the team?
- What are the main issues and challenges in composing a team and working with other team members?

**Version Control**

- What kinds of version control systems are used?
- How often build (or integration) is done?

**Customer**

- Did customers attend any team meetings, if so, what kind of meeting and how often do they attend?
- How do you get feedback from customers?
- How often do you get feedback from customers?
- How do you reflect customer feedback on you project?

**Working environment**

- What is your working environment? (Open environment, cubicle, or personal office)
- What are the pros and cons of your environment?

**Debugging**

- What methods were used for debugging? (Unit testing, system wide testing, etc.)?
- How often do you integrate the changes?
- Who are responsible for testing (developers, Quality Assurance personnel, etc.)?

**Management Tools**

- What tools do you use to manage and track projects?
- How do you share knowledge and skills with other developers?

**Scrum**

- Please describe the pros and cons on each item.

a. Daily Scrum meeting
b. Sprint planning meeting
c. Sprint review meeting
d. Product backlog
e. Sprint backlog
f. Scrum Master

- How often and how long do you have the following meeting?
  a. Daily Scrum meeting
  b. Spring planning meeting
  c. Sprint review meeting

- Who usually attend the following meeting?
  d. Daily Scrum meeting
  e. Spring planning meeting
  f. Sprint review meeting
- Who creates a product backlog and a sprint backlog?
- How do you estimate the time for each task in sprint backlog?
- How much time is allocated to design?
- How much time is allocated to documentation?
- What is Scrum Master's responsibility?

**Unified Process**

- What are the roles of Unified Process in your software development process?
- How does Unified Process help to implement Scrum?
- Are there any issues and challenges in using Scrum with Unified Process?

**General Issues**

- What things did and did not work for you on a project?
- What are the most interesting and unique aspects of Scrum with Unified Process?
- What did you learn, what would you do next time, and what advice do you have for others?

# Interview Questions for Executive Managers

**Project description:**

- What projects were you involved and are you working on?
- What's the nature of project?
- How long does each project take to be completed?

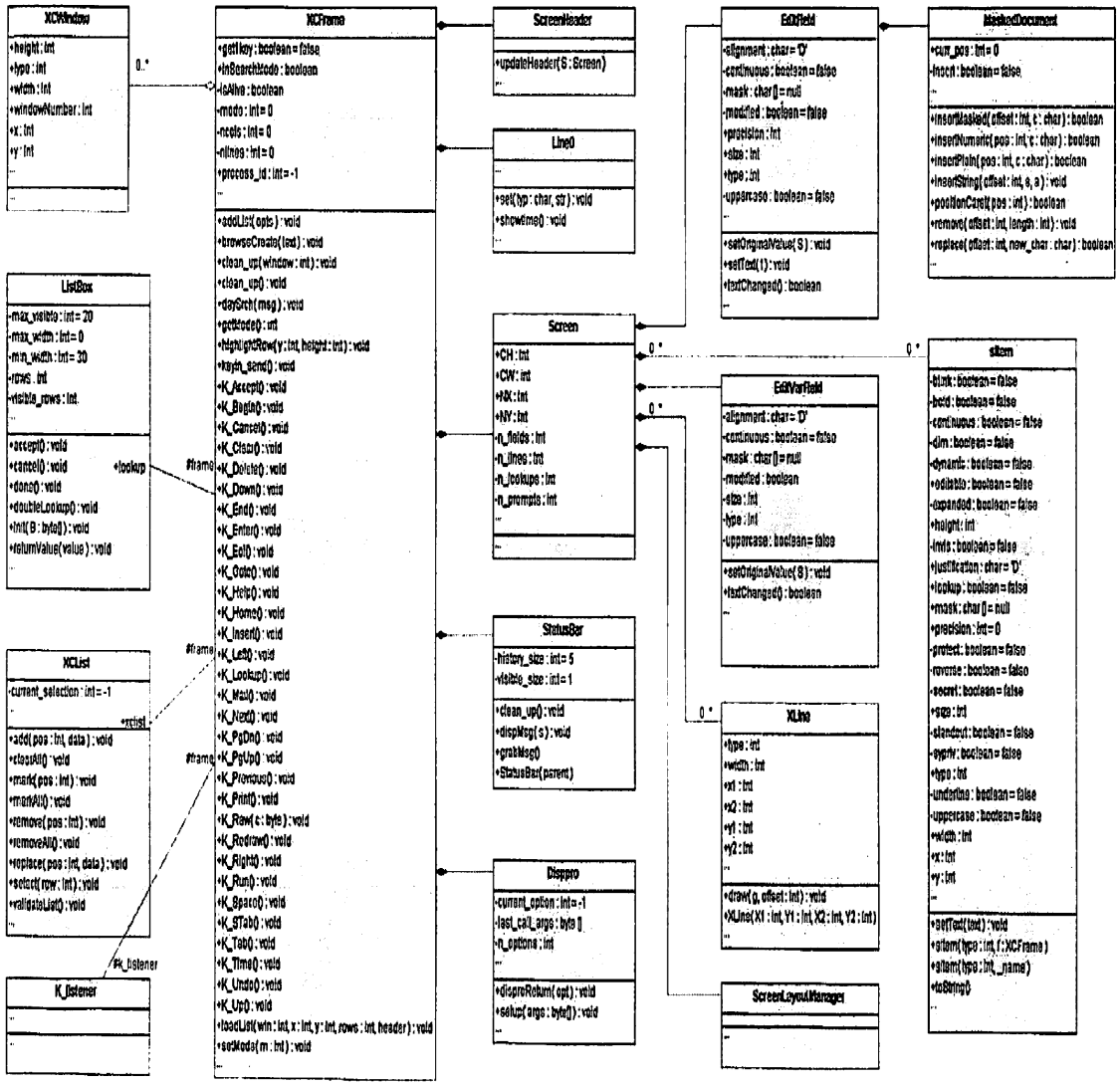
**Bug Rate, Development Time and Cost**

- What is your bug rate in terms of per line of code, per working hours, and per module?
- What is your development time per module and per project?
- What is your development cost per module and per project?
- Compare to a similar project that you completed using traditional method, is you bug rate/development time/development cost increased or decreased?
- What are the factors that attribute to increasing/decreasing bug rate and development time and development cost?

# APPENDIX F. CLASS DIAGRAM

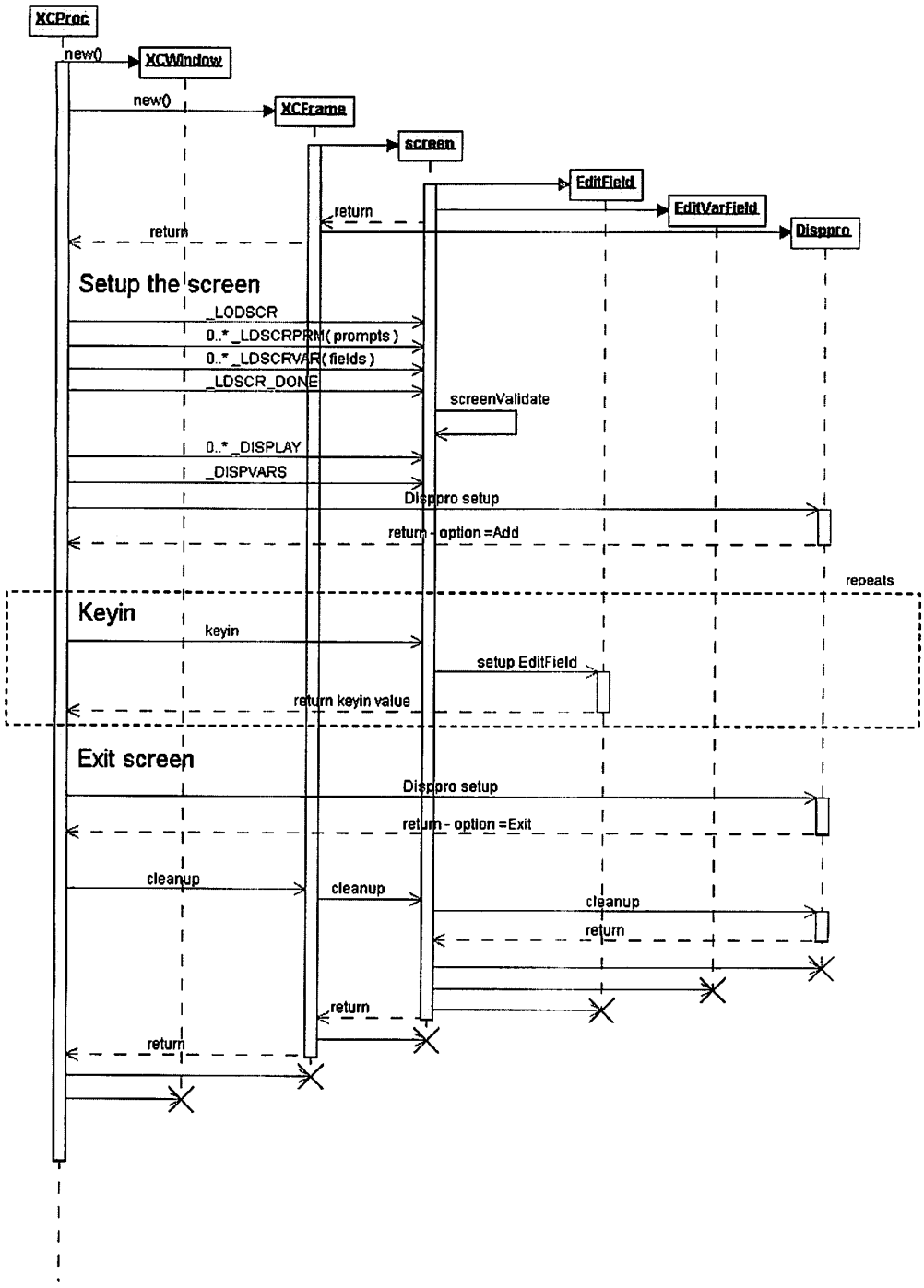
# DIAGRAMS

## Class Diagram



# APPENDIX G. SEQUENCE DIAGRAM

## Sequence Diagram



# APPENDIX H. QUALITATIVE STUDIES

**List of Papers Related to Case Studies:**

| Author | Title | Publisher | Volume/Year/ Page |
|---|---|---|---|
| Walsham, G. and Waema, T. | Information Systems Strategy and Implementation: A Case Study of a Building Society | *ACM Transactions on Information Systems* | (12:2), April 1994, pp. 150-173. |
| Sillince, J.A.A. and Mouakket, S. | Varieties of Political Process During Systems Development," | *Information Systems Research* | (8:4), December 1997, pp. 368-397. |
| Shanks, G. | The Challenges of Strategic Data Planning: an Interpretive Case Study | *Journal of Strategic Information Systems* | 6, 1997, pp. 69-90. |
| Sauer, C. | *Why Information Systems Fail: A Case Study Approach* | Alfred Waller Ltd, Henley-on-Thames | 1993. |
| Robey, D. and Sahay, S. | Transforming Work through Information Technology: A Comparative Case Study of Geographic Information Systems in County Government | *Information Systems Research* | (7:1), 1996, pp. 93-110. |
| Paré, G. | Investigating Information Systems with Positivist Case Study Research | *Communications of the Association for Information Systems* | (13:1), 2004, pp. 233-264. |
| Orlikowski, W.J. | Improvising Organizational Transformation Over Time: A Situated Change Perspective | *Information Systems Research* | (7:1), 1996, pp. 63-92. |
| Markus, M.L. | Finding a Happy Medium: Explaining the Negative Effects of Electronic Communication on Social Life at Work | *ACM Transactions on Information Systems* | 12,2, April 1994, pp. 119-149. |
| Markus, M.L. | Power, Politics and MIS Implementation | *Communications of the ACM* | 26, 1983, pp. 430-444. |
| Manning, P.K. | Information Technology in the Police Context: The "Sailor" Phone | *Information Systems Research* | (7:1), 1996, pp. 52-62. |
| Levine, H.G. and Rossmore, D. | Diagnosing the Human Threats to Information Technology Implementation: A Missing | *Journal of Management Information* | (10:2), Fall 1993, pp. 55-73. |

| | | | |
|---|---|---|---|
| | Factor in Systems Analysis Illustrated in a Case Study | *Systems* | |
| Author | Title | Publisher | Volume/Year/ Page |
| Lee, A.S. | Electronic Mail as a Medium for Rich Communication: An Empirical Investigation Using Hermeneutic Interpretation | *MIS Quarterly* | (18:2), June 1994, pp. 143-157. |
| Kaplan, B. and Duchon, D. | Combining Qualitative and Quantitative Methods in Information Systems Research: A Case Study | *MIS Quarterly* | (12:4) 1988, pp. 571-587. |

**List of Papers Related To Action Research:**

| Author | Title | Publisher | Volume/Year/ Page |
|---|---|---|---|
| Wood Harper, A.T. | Viewpoint: Action Research | *Journal of Information Systems* | (2), 1992, pp.235-236. |
| Warmington, A. | "Action Research: Its Methods and its Implications | *Journal of Applied Systems Analysis* | (7), 1980, pp. 23-39. |
| Street, C.T., and Meister, D.B. | Small Business Growth And Internal Transparency: The Role Of Information Systems | *MIS Quarterly* | (28:3) 2004, pp 473-506. |
| Mårtensson, P., and Lee, A.S. | Dialogical Action Research At Omega Corporation | *MIS Quarterly* | (28:3) 2004, pp 507-536. |
| Mansell, G. | Action research in information systems development | *Journal of Information Systems* | (1), 1991, pp. 29-40. |
| Lindgren, R., Henfridsson, O., and Schultze, U. | Design Principles For Competence Management Systems: A Synthesis Of An Action Research Study | *MIS Quarterly* | (28:3) 2004, pp 435-472. |
| Lee, A.S., Baskerville, R.L. and Davies, L. | A Workshop on Two Techniques for Qualitative Data Analysis: Action Research and Ethnography | *Proceedings of the Thirteenth International Conference on Information Systems* | 1992, p. 305-306. |
| Kohli, R., and Kettinger, W.J. | Informating The Clan: Controlling Physicians' Costs And Outcomes | *MIS Quarterly* | (28:3) 2004, pp 363-394 |
| Markus, M.L. | Power, Politics and MIS Implementation | *Communications of the ACM* | 26, 1983, pp. 430-444. |
| Kock, N.F., Jr., McQueen, R.J. and Scott, J.L. | Can Action Research be Made More Rigorous in a Positivist Sense? The Contribution of an Iterative Approach | *Journal of Systems and Information Technology* | (1:1), 1997, pp. 1-24. |

**List of Papers Related To Ethnographic Research:**

| Author | Title | Publisher | Volume/Year/ Page |
|---|---|---|---|
| Suchman, L. | Making Work Visible | *Communications of the ACM* | (38:9), 1995, pp. 56-64. |
| Simonsen, J. and Kensing, F. | Using Ethnography in Contextual Design | *Communications of the ACM* | (40:7), 1997, pp. 82-88. |
| Randall, D., Hughes, J. and Shapiro, D. | Steps towards a partnership: Ethnography and System Design. In M. Jirotka & J. Goguen (Eds.), *Requirements Engineering: Social and Technical Issues.* | London, Academic Press | 1994. |
| Prasad, P. | Systems of meaning: ethnography as a methodology for the study of information technologies. In A.S. Lee, J. Liebenau, & J.I. De Gross (Eds.), *Information Systems and Qualitative Research.* | Chapman and Hall, London | 1997, pp. 101-118. |
| Ngwenyama, O.K., Harvey, L., Myers, M.D. and Wynn, E. | Ethnographic Research in Information Systems: An Exploration of Three Alternative Approaches to Ethnography. | *Proceedings of the Eighteenth International Conference on Information Systems* | 14-17 December, 1997. |
| Myers, M.D. and Young, L. W. | Hidden Agendas, Power, and Managerial Assumptions in Information Systems Development: An Ethnographic Study. | *Information Technology & People* | (10:3), pp. 224-240. |
| Myers, M.D. | Critical Ethnography in Information Systems. In A.S. Lee, J. Liebenau and J.I. DeGross (Eds.), *Information Systems and Qualitative Research.* | Chapman and Hall, London, | 1997, pp. 276-300. |
| Klein, H. K. and Myers, M.D. | A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems. | *MIS Quarterly, Special Issue on Intensive Research* | (23:1), 1999, pp. 67-93. |
| Harvey, L. and Myers, M.D. | Scholarship and practice: the contribution of ethnographic | *Information Technology &* | (8:3), 1995, pp. 13-27. |

| | | | |
|---|---|---|---|
| | research methods to bridging the gap. | *People* | |
| Author | Title | Publisher | Volume/Year/ Page |
| Benson, D.H. | A field study of end-user computing: Findings and issues. | *MIS Quarterly* | December, 1983, pp. 35-45. |

**List of Papers Related To Grounded Theory:**

| Author | Title | Publisher | Volume/Year/ Page |
|---|---|---|---|
| Urquhart, C. | An Encounter with Grounded Theory: Tackling the Practical and Philosophical Issues. In E. Trauth (Ed.), *Qualitative Research in IS: Issues and Trends*. | Idea Group Publishing, Hershey | 2001, pp. 104-140. |
| Urquhart, C. | Exploring Analyst-Client Communication: Using Grounded Theory Techniques to Investigate Interaction in Informal Requirements Gathering. In A.S. Lee, J. Liebenau, & J.I. DeGross (Eds.), *Information Systems and Qualitative Research*. | Chapman and Hall, London | 1997, pp. 149-181. |
| Smit, J. | Grounded Theory Methodology in IS Research: Glaser versus Strauss | *South African Computer Journal* | (24:November), 1999, pp. 219-222. |
| PriesHeje, J. | Three barriers for continuing use of computerbased tools: a grounded theory approach | *Scandinavian Journal of Information Systems* | (4), 1992, pp. 119-136. |
| Pettigrew, A.M. | Contextualist Research and the Study of Organizational Change Processes. In E. Mumford, R. Hirschheim, G. Fitzgerald, & A.T. Wood-Harper (Eds.), *Research Methods in Information Systems*. | Amsterdam, North Holland | 1985, pp. 53-78 |
| Pandit, Maresh R. ",", | The Creation of Theory: A Recent Application of the Grounded Theory Method | *The Qualitative Report* | 2(4), 1996 |
| Pace, S. | A grounded theory of the flow experiences of Web users | International Journal of Human-Computer Studies | (60:3) 2004, pp 327-363. |
| Hughes, J. and D, H. | Grounded Theory: Never Knowingly Understood | *Information Systems Review* | (1), 2000, pp. 181-197. |
| De Vreede, | Exploring the Application and | *Journal of* | (15:3), 1999, pp. |

| | | | |
|---|---|---|---|
| G.J., Jones, N. and Mgaya, R. | Acceptance of Group Support Systems in Africa | *Management Information Systems* | 197-212. |
| Author | Title | Publisher | Volume/Year/ Page |
| Bowker, G., Timmermans, S. and Star, S.L. | Infrastructure and Organizational Transformation: Classifying Nurses' Work. In W. Orlikowski, G. Walsham, M. Jones, & J.D. DeGross (Eds.), *Information Technology and Changes in Organizational Work.* | London: Chapman and Hall | 1995, pp. 344-370. |

# CURRICULUM VITAE

Juyun Joey Cho

Assistant Professor
Computer Information Systems
Colorado State University – Pueblo
2200 Bonforte Blvd.
Pueblo, CO 81001
joey.cho@colostate-pueblo.edu

## EDUCATION

- **Ph.D. Management Information Systems**, (2009), Utah State University.
- **Second Master of Computer Science** (2000), Utah State University.
- **Second Bachelor of Computer Science** (1995), Utah State University.
- **Master of Compute**r **Engineering** (1990), Chungbuk National University, Korea.
- **Bachelor of Computer Engineering,** (1988), Chungbuk National University, Korea.

## REFEREED JOURNAL ARTICLES

- **Cho, J.** (2009). A hybrid software development method for large-scale projects: rational unified process with Scrum. *Issues in Information Systems, X(2),* 340-348.

- Cardon, P., Marshall, B., Norris, D., **Cho, J.**, et al. (2009). Online and offline social ties of social network website users: an exploratory study in eleven societies. *Journal of Computer Information Systems, 50*(1), 54-64.

- **Cho, J.** (2008). Issues and challenges of agile software development with scrum. *Issues in Information Systems, IX(2),*188-195.

- **Cho, J.,** Jones, S., & Olsen, D. (2008). An exploratory study on factors influencing major selection. *Issues in Information Systems, IX(1),*168-175.

- **Cho, J.** (2007). Globalization and global software development. *Issues in Information Systems, VIII*(2), 287-290.

## CONFERENCE PPROCEEDINGS

- **Cho, J.**, Marjanovi, B., Chi, T., Vogl, M., & Bechtoldt, C. (2009). What can Yahoo! do to be more competitive?. *Proceedings of 8th ISOneWorld International Conference*, Las Vegas, Nevada, 66: 1-12

- **Cho, J.** (2007). Distributed Scrum for large-scale and mission-critical projects. *Proceedings of 13th Americas Conference on Information Systems (AMCIS-07)*, Keystone, Colorado, amcis-508-2007 (8 pages).

- **Cho, J.**, Couraud, J., & Olsen, D. (2007). IS curriculum recommendations for web courses based on current technology use by fortune 400 companies. *Proceedings of 13th Americas Conference on Information Systems (AMCIS-07)*, Keystone, Colorado, amcis-506-2007 (10 pages).

- **Cho, J.** (2007). An exploratory study on management of virtual teams in distributed scrum software development method for large-scale and complex projects. *Proceedings of 13th Americas Conference on Information Systems (AMCIS-07)*, Keystone, Colorado (submitted to Doctoral Consortium).

- Marshall, B., **Cho, J.,** & Harris, M. (2006). Application of recent trends in web technologies. *Proceedings of 46th International Association for Computer Information Systems (IACIS-06)*, Reno, Las Vegas, 34 (abstract only).

- **Cho, J.,** Kim, Y., & Olsen, D. (2006). A case study on the applicability and effectiveness of scrum software development in mission-critical and large-scale projects. *Proceedings of 12th Americas Conference on Information Systems (AMCIS-06)*, Acapulco, Mexico, 3705-3711.

**TECHNICAL PAPERS**

- Cho, J. (2000)
  "An Interactive Computer and Literacy Instruction System" (**Funded by NSF & U.S. Department of Education**)
  Master's report, Computer Science, Utah State University, 2000.

- Cho, J. (1990)
  "A Study on the Optimal Fixed-Size Window Scheme in RISC (Reduced Instruction Set Computer) system"
  Master's thesis, Computer Engineering, Chungbuk National University, 1990.

**CONFERENCE PRESENTATIONS**

- "A Hybrid Software Development Method for Large-Scale Projects: Rational Unified Process with Scrum", *the 49th International Association for Computer Information Systems (IACIS)*, Pittsburgh, Pennsylvania, 2009.

- "What can Yahoo! do to be more Competitive?", *the 8th of ISOneWorld International Conference*, Las Vegas, Nevada, 2009.

- "Issues and Challenges of Agile Software Development with Scrum", *the 48th International Association for Computer Information Systems (IACIS)*, Savannah, Georgia, 2008

- "An Exploratory Study on Factors Influencing Major Selection", *the 48th International Association for Computer Information Systems (IACIS)*, Savannah, Georgia, 2008

- "Globalization and Global Software Development", *the 47th International Association for Computer Information Systems (IACIS)*, Vancouver, Canada, 2007.

- "Distributed Scrum for Large-Scale and Mission-Critical Projects", *the 13th Americas Conference on Information Systems (AMCIS-07)*, Keystone, Colorado, 2007.

- "IS Curriculum Recommendations for Web Courses Based on Current Technology Use By Fortune 400 Companies", *Proceedings of 13th Americas Conference on Information Systems (AMCIS-07)*, Keystone, Colorado, 2007

- "An Exploratory Study on Management of Virtual Teams in Distributed Scrum Software Development Method for Large-Scale and Complex Projects", *the 13th Americas Conference on Information Systems (AMCIS-07)*, Keystone, Colorado, 2007.

- "Agile Software Development Methods in Distributed Software Development Environment", *10th Annual Graduate Research Symposium*, Utah State University, 2007.

- "A Case Study on the Applicability and Effectiveness of Scrum Software Development in Mission-Critical and Large-Scale Projects", *the 12th Americas Conference on Information Systems (AMCIS-06)*, Acapulco, Mexico, 2006.

- "Agile Software Development Methods for Large-Scale Projects", *9th Annual Graduate Research Symposium*, Utah State University, 2006.

**TEACHING EXPERIENCE**

Colorado State University - Pueblo, Colorado

- **MGMT 565** Management Information Systems, spring 2009
- **CIS 411** Internet Server Side Programming, spring 2009
- **CIS 311** Introduction to Web Development, fall 2008
- **CIS 271** Advanced Program Design with Java, fall 2008, 2009
- **CIS 171** Introduction to Java Programming, fall 2008, 2009
- **MGMT 365** Introduction to MIS, summer 2008, 2009
- **BUSAD 265** Inferential Statistics, summer 2008, 2009
- **CIS 240** Object Oriented Analysis and Design, spring 2008, 2009, fall 2009
- **CIS 4/561** IT Security Management, spring 2008

Utah State University, Logan, Utah

- **BUS 3510** Business Programming (Satellite distance education), spring 2007

- **BIS 5450/6450** Designing Graphical User Interfaces for E-Commerce, spring 2007
- **BIS 2100** Principles of Management Information Systems, fall 2006 & fall 2007 (two sections)
- **BIS 5450/6450** Designing Graphical User Interfaces for E-Commerce, fall 2006 (*Real-life project for RCF Inc.*)
- **BIS 6500** Developing Business Information Systems with Advanced Software Concepts (Satellite distance education), summer 2006
- **BIS 3500/6500** Management Information Systems Development, fall 2005

**PROFESSIONAL ACTIVITIES & AFFILIATIONS**

- **Reviewer**
  - Issues in Information Systems, 2007,2008, 2009
  - Americas Conference on Information Systems (AMCIS) 2007
  - Pacific Asia Conference on Information Systems (PACIS) 2007
  - International Association for Computer Information Systems (IACIS) 2007, 2008
  - AMCIS Doctoral Consortium, 2007

- **Track Chair:**
  - Systems Analysis and Design, ISOneWorld 2009, Las Vegas, USA, 2009.
  - Systems Analysis and Design, ISOneWorld 2008, Las Vegas, USA, 2008.

- **Session Chair**:
  - Software Engineering, International Association for Computer Information Systems (IACIS), Savannah, Georgia, 2008.
  - Expert systems, International Association for Computer Information Systems (IACIS), Vancouver, Canada, 2007.

- **Student Representative** for Doctoral Advisory Committee in Interdisciplinary Department Program, Utah State University, 2006-2007.

- **Professional Membership**
  - International Association for Computer Information Systems (2006-present)
  - Association for Information Systems (2005-2007)
  - ACM Special Interest Group on Knowledge Discovery & Data Mining (SIGKDD) 2003-2008.

**HONORS & REWORDS**

- Faculty Development Grant, Colorado State University-Pueblo, 2008 & 2009.
- Accepted to AMCIS 2007 Doctoral Consortium.

- Travel Funds Award, Graduate Student Senate (GSS), Utah State University, 2006 & 2007.
- Travel Funds Award, Management Information Systems Dept, Utah State University, 2006.
- Korean President's Scholarship (IL-HAE scholarship foundation, full-tuition & living expenses), 1985-1987.
- Chungbuk National University President's Scholarship (Full tuition), 1984.

**WORK EXPERIENCE**

- **Assistant Professor, January 2008-Present**
  - Colorado State University-Pueblo, 2200 Bonforte Blvd, Pueblo, CO 81001

- **Graduate Instructor, September 2005-September, 2007**
  - Utah State University, Logan, UT 84322

- **Software Engineer II, December 2003-January, 2006**
  Spillman Technologies Inc., 4625 West Lake Park Blvd. Salt Lake City, UT 84120 (*provides complicated public safety software solutions such as 911 dispatch system, jail management system, and fire/emergency medical system*. http://www.spillman.com/)
  - Performs product design and system analysis.
  - Researches, develops, designs and maintains application software.
  - Establishes standards, procedures, and specification for software development and maintenance.
  - Implements changes, corrects problems to modify, develop, and enhance functionality.
  - Analyzes software requirement to determine feasibility of design.
  - Consults with customer concerning design issues.
  - Organizes and participates in software design and code reviews.
  - Participates in the definition of technical requirements, software procedures, software installation, upgrade issues, and user documentation for application software.
  - Performs code verifications, release testing, and beta support for assigned products.
  - Researches problems discovered by Quality Assurance for product support.
  - Develops solutions to problems.
  - Researches and understands the marketing requirements for a product, including target environment, performance criteria and competitive issues.

- **Software Engineer II/Project Manager, February 2000-September, 2003**

Global Mart / PickSend Inc., 517 W. 100 N. Providence, UT 84332 (*E-Commerce retail/whole sale company selling products through an internet* http://www.picksend.com )

- Supervised 4-6 web programmers.
- Implemented GUI using Borland C++ Builder for order processing system, product management system, purchasing system, receiving system, inventory management system, credit card transaction system, accounting system, retail store management system (Cash sales, layaway and rental system), vendor management system, and RMA system.
- Set up interface between our order processing system and remote site drop shippers database system using ODBC.
- Implemented database-enabled Shopping Cart and Cashpad system using CGI for globalmart.com, escapeoutdoors.com, globalmartoutlet.com, and poweredmarket.com.
- Implemented Server Monitor system using Client-Server TCP/IP Socket program between Windows and Linux Machines. (When server goes down Windows machine calls out manager's cell phone)
- Implemented Interface between our database and UPS/Fedex shipping system using ODBC in Windows NT system.
- Implemented various reporting systems using Perl.
- Designed Database driven dynamic web site using Perl , Perl DBI, and PHP. (http://www.globalmart.com, http://www.picksend.com)
- Implemented cron jobs for automated ftp and data dump using Perl.
- Set up search engine strategies for company's web site.
- Implemented Linux Raid System.
- Set up Web Statistics Program.
- Set up and Maintained Linux Gateway Server(Sangome Card for T1 frame relay server), Firewall Server(IPCHAINS), Proxy Server, Email Server, Apache Web Server, Fax Server, DNS Server, File/Printer Server, Database Server(MySQL), Backup Server.

- **Software Engineer II, September 1999-Feburary 2000**
  Intermountain Transcription Services Inc., 570 Research Parkway, North Logan, UT 84341 (*provides medical transcription services*)
  Designed, implemented, tested, and debugged the following
  - Object Oriented Windows Program using Borland C++ Builder.
  - Relational Database System.
  - GUI for medical transcription system.
  - Set up/Maintain Linux Server.

- **Software Engineer, September 1997-September 1999**
  Digitran System Inc., 2176N. Main, Logan, UT (*makes sophisticated computer simulated software for mining, petroleum, vehicle, and maritime crane industries*)
  Analyzed, designed, implemented, maintained the following

- TCP/IP, UDP/IP Networking, Low Level Socket Networking, and Serial Communication for Win95/NT and UNIX.
- Graphical User Interface for Win95/NT and X Motif for SGI IRIX/LINUX using MS visual C++ and Dataview Xdesigner.
- Motion, Hardware Interfacing.
- *Simulation Logic for Crane, Oil, and Truck Simulator.*
- Multi-Channel Sound System Control for Win95 and Linux.
- OpenGL Graphics Program.
- Converting code from SGI IRIX to Linux.
- Set up Email/Web Server, LAN, CVS(Concurrent Version System) and Firewalls.

- **Research Assistant in Multimedia Lab, June 1996-September 1997**
  Computer Science Department, Utah State University, Logan, UT
  - Participated in the research project called “Computer Literacy” -- A multi-media computer based instructional system that teaches college student how to use Word Processor, Spread Sheet, VAX/VMS Email, Windows95 Systems, UNIX Systems and Internet. (ToolBook and Paradox database were used, **Funded by National Science Foundation & the United States Department of Education**)

- **Teaching Assistant, March 1996-June 1996**
  Computer Science Department, Utah State University, Logan, UT
  - Helped students with C and C++ program problems.
  - Graded CS410. (Operating Systems)

- **Research Assistant, September 1995-March 1996**
  Computational Sciences Division, Space Dynamics Lab, Logan, UT
  - Tested software and data of **NASA projects** using IRIS "Explorer" in SGI workstation.

- **Assistant Technician, October 1994-September 1995**
  Merrill Library, Utah State University, Logan, UT
  - Dumped data and maintained MDAS of Merrill & Sci-Tech Libraries.
  - Retrieved and analyzed statistical data to see how the Libraries databases are being accessed and used.

- **Teaching Assistant, 1988-1990**
  Dept. of Computer Engineering, Chungbuk National University, Chungbuk, Korea.
  - Taught architecture of Z-80 micro processor and Assembly Language.
  - Helped students build Analogue/Digital Converter.

- **Computer Lab Consultant, March 1988-February 1989**

Dept. of Computer Engineering, Chungbuk National University, Chungbuk, Korea.

- Maintained Software program and hardware equipment.
- Helped students with computer problems.

**PROFESSIONAL SKILLS**

- **Computer Programming Languages:**
  C/C++, C#, PHP, Java, Java Script, Perl script, Perl DBI , UNIX shell script, HTML, PHP, PVM, Lisp, Prolog, Assembly Language, Pascal, FORTRAN, BASIC, Visual Basic.

- **Software Packages:**
  MS Visual Studio .NET, MS Visual C++/J++, Borland C++ Builder, dBaseIV, Paradox, MySQL, Asymmetric Toolbook, Matlab, NeualWare, SLAM II.

- **Operating Systems:**
  Windows3.1/95/98/2000/XP/NT, Linux, SunOS, IBM AIX, HP-UX, SGI-IRIX, VAX/VMS, DEC ULTRIX, MS-DOS.

- **Platforms:**
  HP-UX 9000/700, SunSPARC, SGI, DEC 5000, VAX 11/780, IBM PC/Compatibles, Macintosh.